VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

Elektronikos fakultetas

AUTOMATIKOS KATEDRA

**Jaroslavas Semaško**

# ROBOTO VALDYMO SISTEMOS OPTIMIZAVIMAS

# OPTIMIZATION OF ROBOT CONTROL SYSTEM

**Baigiamasis magistro darbas**

Automatikos studijų programa

Automatinių sistemų

elektros ir elektronikos inžinerija

Vadovas prof. habil. dr. A. Dambrauskas

Vilnius, 2004

Vilniaus gedimimo technikos universitetas
Elektronikos fakultetas
Automatikos katedra

ISBN ISSN
Egz. sk.
Data

Magistratūros baigiamasis darbas (tezės)

Pavadinimas: Roboto valdymo sistemos optimizavimas

Autorius: Jaroslavas Semaško, gr. AUSm-2

Kalba

☐ Lietuvių

☐ Užsienio

## ANOTACIJA

Baigiamajame darbe aprašomi robotų valdymo optimizavimo metodai, algoritmai ir būdai.

Analitinėje dalyje nagrinėjamos robotų automatinės valdymo sistemos ir robotų valdymas. Taip pat aptartas roboto ir mikrovaldiklio suderinimas ir paieškos optimizavimo uždaviniai.

Pagrindinis darbo tikslas – panaudojant optimizavimo metodus ir būdus optimizuoti roboto valdymo sistemą, darbe pateikti robotų valdymo optimizavimo algoritmai. Optimizuotas roboto URTK valdymo algoritmas ir sudaryta nauja robotų valdymo paprogramė. Programa Kvaziol, taikant algoritminius sintezės metodus, optimizuotas atviros kvazioptimalios valdymo sistemos pereinamasis procesas. Pagal darbe padarytą analizę ir rezultatus parašytos išvados ir pasiūlymai.

.

# Turinys

# 1. ĮVADAS

Optimalių sprendimų paieška optimizavimo metodais tampa įprastu tyrimu, sprendžiant projektavimo, mokslo ir gamybos uždavinius. Vis dažniau taikomi paieškos optimizavimo metodai, kurie suteikia galimybę spręsti ekstremalaus valdymo uždavinius, esant neapibrėžtoms sąlygoms. Tokios sąlygos atsiranda, kai reikia derinti naują technologinį procesą ir nėra optimizuojamo rodiklio analitinės priklausomybės nuo valdymų kintamųjų ir negalima tiksliai išmatuoti objekto išėjimo dydžių. Atliekant paieškos eksperimentus, gaunama papildoma informacija apie objektą. Tai paieškos optimizavimo metodams suteikia universalumo. Šie metodai sėkmingai taikomi technologiniams procesams optimizuoti, konstravimo, planavimo ir projektavimo uždaviniams spręsti, naujų medžiagų sintezėje, eksperimentiniuose tyrimuose.

Robotų valdymo optimizavimas labai padidina jų panaudojimo efektyvumą. Problemų ir optimalaus valdymo metodų sprendimai skirtingų tipų robotams, nors ir turi savo specifiką, bet daugeliu atvejų labai panašūs. Todėl iškilusias problemas racionalu nagrinėti kartu laikantis vienų metodologijos pozicijų. Toks svarstymas tinkamas dar ir todėl, kad suteikia galimybę pasidalyti patirtimi panaudojant vieno tipo robotų optimizavimo valdymą kitiems.

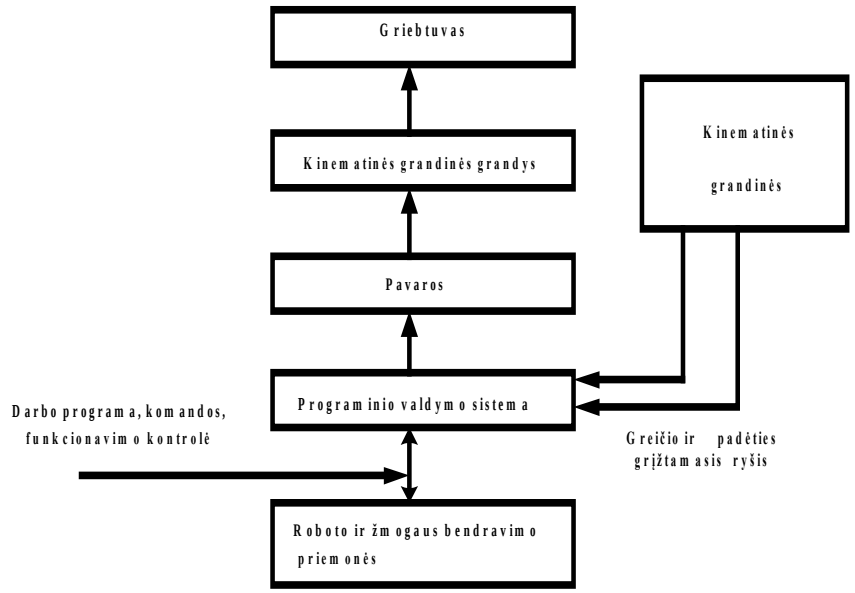
Optimizuojant roboto judėjimą, galima siekti įvairių tikslų. Jeigu valdymo sistema hierarchinė, tai tenka kalbėti apie įvairių hierarchinės sistemos lygių optimizavimą: judesio planavimo ir programavimo lygio arba apie optimizavimą stabilizacijos ir adaptacijos proceso metu. Darbe apžvelgti visų minėtų lygių optimizavimo uždaviniai ir metodai.

# 2. ANALITINĖ APŽVALGA

## 2.1. Robotų automatinės valdymo sistemos

Pagrindinis robotų automatinių valdymo sistemų ypatumas - nepriklausomas darbas dalyvaujant žmogui. Operatorius dalyvauja tik tada, kai reikia robotą mokyti, paleisti ir periodiškai stebėti jo veikimą. Valdymo dėsniai automatinėse valdymo sistemose yra realizuoti kaip valdymo programos. Todėl galimi įvairūs darbo režimai pasirinktai programai atlikti: nekeičiamas programos valdymas (dar vadinamas pagrindine programa), kai proceso metu negalima jo reguliuoti; adaptyvus programos keitimas, atsižvelgiant į darbo sąlygas ir aplinką; valdymas darbo metu, nors ir nėra tiksliai pasirinktos programos.

Pagrindinės programos valdymas būdingas pirmosios kartos robotams. Tai gana paprastas valdymo būdas, ypač realizuojant atvirojo ciklo valdymą. Sudėtingesniais atvejais naudojami uždarojo tipo ciklai. Pirmosios kartos robotas aptartas šiame darbe. Tokio roboto struktūrinė schema yra pateikta **2.1.1 pav**.




**2.1.1 pav. Pirmosios kartos pramoninių robotų struktūrinė schema**

Sistemos sugebėjimas reaguoti į aplinkos pokyčius, įgalina robotą programos vykdymo metu prisitaikyti prie aplinkos. Šie robotai priskiriami prie antrosios kartos robotų, dar vadinamų adaptyviaisiais robotais. Jie turi jutiklius, kurie teikia informaciją apie išorinę aplinką.

Trečiajai kartai galima priskirti intelektinius robotus. Jie sugeba atpažinti aplinką, reaguoti į pašalinius veiksnius, priima sprendimus savarankiškai ir keičia elgesį atlikdami nustatytą užduotį. Trečiosios kartos robotai sugeba save apmokyti, todėl jų informacijos apimtis yra gana didelė, palyginti su antrosios kartos robotais. Šiame procese operatorius nustato tik galutinį gamybinio proceso tikslą, o visi sprendimai priimami automatiškai be žmogaus **[17].**

Pagal atliekamų operacijų pobūdį visi robotai skirstomi į technologinius ir pagalbinius. Pagal valdymo tipą skiriami:

1) ciklinio programinio valdymo;
2) pozicinio programinio valdymo;
3) kontūrinio programinio valdymo;
4) adaptyviojo programinio valdymo.

Šiame darbe yra panaudotas pozicinio programinio valdymo robotas, nes įrenginio darbo organas valdomas pagal nurodytus pozicionavimo taškus, nekontroliuojant jo judesio tarp taškų.

Pagal programavimo būdą galima priskirti prie programuojamų apmokamų robotų, t.y. valdymo programa sudaroma ir įvedama operatoriaus, atliekant darbo organo parengiamuosius judesius ir juos užrašant į valdymo įrenginį.

## 2.2. Roboto valdymo analizė

Atsižvelgiant į objekto sudėtingumą ir technologinio proceso pobūdį, tikslinga manipuliatoriaus valdymui panaudoti programuojamąjį mikrovaldiklį.

Bet kuris valdiklis yra specializuotas kompiuteris, sukurtas technologinių procesų valdyti realiuoju laiku, turintis skaitmeninio apdorojimo galimybę. Pagrindinės programuojamojo valdiklio funkcijos yra sekti prijungtų prie jo įrenginių būseną, apdoroti vartotojo programą ir veikti valdomuosius įrenginius formuojant jiems atitinkamus valdymo signalus.

Įvairių rūšių valdiklius gamina kompanija Modicon. Jos siūlomi programuojamieji valdikliai, nesvarbu kokia jų taikymo sritis, pagrįsti tais pačiais darbo principais: visi jie programuojami relinių simbolių kalba (RSK), turi bendrą rinkinį komandų, realizuojančių aritmetines bei logines operacijas, duomenų persiuntimą, darbą su matricomis ir kitas specialias funkcijas. Didelė gaminamų Modicon valdiklių įvairovė leidžia vartotojui pasirinkti reikiamą valdiklių tipą ir jų struktūrą, atsižvelgiant į kuriamos automatizavimo sistemos sudėtingumą, našumą ir aparatinį suderinimą.

Didelis Modicon valdiklių privalumas – tai visiškas suderinamumas tarpusavyje, nesvarbu koks jų dydis ir konstrukcija. Valdiklių sisteminėje programinėje įrangoje naudojamos bendros visų tipų valdikliams programavimo funkcijos. Tai reiškia, kad vartotojo programa, sukurta mažos galios valdikliui, gali būti perkelta į didelės galios valdiklį arba atvirkščiai. Automatizavimo uždaviniui tampant sudėtingam, vis vien galima pakeisti mažos galios valdiklį didesniu neperrašant vartotojo programų.

984 modelio mikrovaldikliai išleidžiami keturių aparatinių tipų:

1) dideli, turintys savo stovą;

2) vidutiniai, esantys pirmame 800 įvedimo/išvedimo serijos modulių lizde;

3) įmontuojami kaip pagrindinio kompiuterio plėtinys;

4) pigūs ir lengvai įrengiami, kompaktiški valdikliai, darbui su aparatūra, kuriai nekeliami griežti reikalavimai.

Reikšmingas šio modelio valdiklių privalumas yra jų tarpusavio suderinamumas, nesvarbu kokia galia ir aparatinė įranga. Taigi viena vartotojo programa, sukurta, pavyzdžiui, galingam valdikliui, gali būti perkelta į mažą ir atvirkščiai. Todėl labai sumažėja įsisavinimo ir apmokymo išlaidos,

nes vartotojas, susipažinęs su vienu valdikliu, gali lengvai įsiminti darbo su kitu valdikliu ypatumus **[14].** Firmos Modicon gaminamų valdiklių charakteristikos ir parametrai pateikti **2.2.1** lentelėje.

**2.2.1 lentelė. 984 modelio mikrovaldiklių parametrai**

| Markė | Konstrukcija | Greitaveika ms/Kžd | Skilčių skaičius | Atmintis skirta logikai | Būsenos atmintis | Būsenos atmintis |
|---|---|---|---|---|---|---|
| 984 B | Stove | 0,75 | 24 | 32/64 | 9999 | 8192 |
| 984-780/ | Lizde | 1,5 | 24 | 16/32 | 9999 | 8192 |
| 984 A | Stove | 2,0 | 24 | 12 | 9999 | 8192 |
| Q984 | Įmontuojamas | 0,75 | 16 | 16/32 | 1920 | 2048 |
| 984 X | Stove | 0,75 | 16 | 8 | 4096 | 2048 |
| 984-685 | Lizde | 2,0 | 16 | 8//16 | 4096 | 2048 |
| 984-680 | Lizde | 3,0 | 16 | 8/16 | 1920 | 2048 |
| AT-984 | Įmontuojamas | 1,5 | 16 | 8 | 1920 | 2048 |
| MC-984 | Įmontuojamas | 1,5 | 16 | 8 | 1920 | 2048 |
| 984-485 | Lizde | 3,0 | 16 | 4/8 | 1920 | 2048 |
| 984-480 | Lizde | 5,0 | 16 | 4/8 | 1920 | 2048 |
| 984-385 | Lizde | 3,0 | 16 | 4/8 | 1920 | 2048 |
| 984-381 | Lizde | 5,0 | 16 | 1,5/4/6 | 1920 | 2048 |
| 984-380 | Lizde | 5,0 | 16 | 1,5/4/6 | 1920 | 2048 |
| 984-145 | Kompaktiškas | 4,25 | 16 | 8 | 1920 | 2048 |
| 984-130 | Kompaktiškas | 4,25 | 16 | 4 | 1920 | 2048 |
| 984-120 | Kompaktiškas | 4,25 | 16 | 1,5 | 1920 | 2048 |
| Micro-984 | Miniatiūrinis | 5 | 16 | 4 | 1920 | 2048 |

Visi 984 modelio valdikliai turi bendrą architektūrą, į kurią įeina:

1) atmintinė, kurioje saugoma:

a) operatyvioji vartotojo programos ir informacijos apie valdiklio ir įrenginių būsena;

b) KMOP tipo su bateriniu maitinimu sisteminiai parametrai;

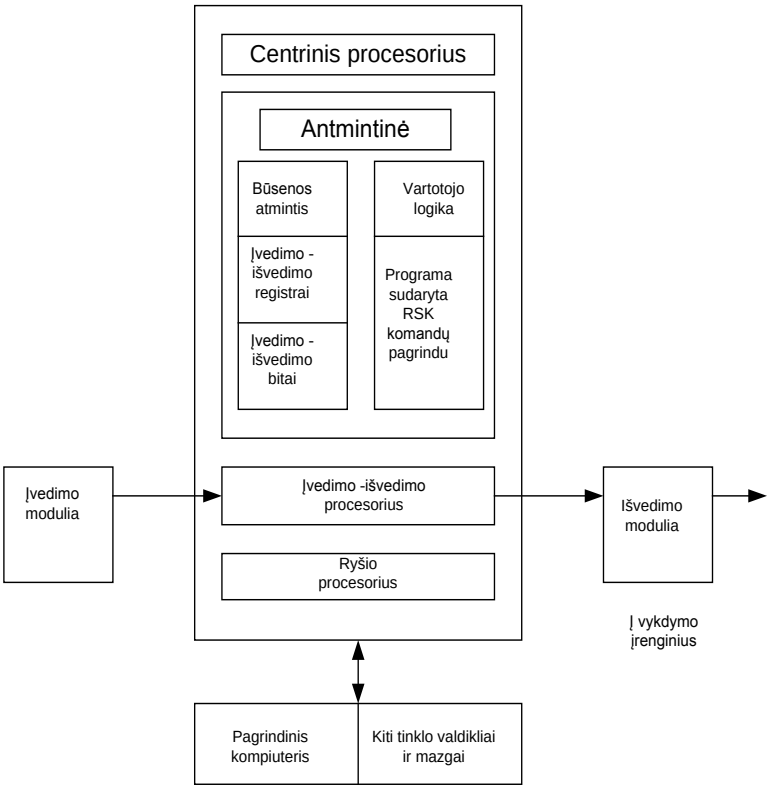
c) pastovi valdymo atmintinė programoms saugoti;

2) procesorius , apdorojantis vartotojo programą;

3) įvedimo/išvedimo signalų apdorojimo blokas, valdantis signalų persiuntimą iš Į/I modulių į būsenos atmintį ir atgal;

4) priėmimo/perdavimo blokas, sudarytas iš vieno ar kelių sąsajos prievadų, kuriais valdiklis sujungiamas su programavimo plokštėmis, pagrindiniu kompiuteriu ir kitais valdikliais ar vietinio duomenų perdavimo tinklo mazgais;

Tokio tipo architektūra parodytas **2.2.2 pav. [11]**:




**2.2.2 pav. 984 modelio valdiklio architektūra**

Atsižvelgiant į techninius duomenis, matyti, kad geriausiai reikalavimus atitinka modelis Micro-984. Pagrindiniai šio valdiklio privalumai - miniatiūrinė aparatinė įranga ir patogus naudoti. Iš trūkumų galima paminėti mažą loginės atminties apimtį 4 Kbaitai ir palyginti lėtą veikimą - 5.0ms/K žodžiui, tačiau konkrečiu atveju į šiuos trūkumus galima neatsižvelgti. Šis mikrovaldiklis maitinamas iš 220V tinklo. Maitinimui prijungti valdiklio viršutinėje plokštėje numatyti penki gnybtai. Prie gnybtų, pažymėtų 1 ir 4 skaitmenimis, jungiamas maitinimo laidas, o 2 ir 3 gnybtai trumpinami trumpikliu. 5 gnybtas naudojamas prietaisui įžeminti.

Valdiklio fizinių įėjimų ir išėjimų būsenai kontroliuoti naudojami 28 šviesos diodai, kurių kiekvienas turi tą patį numerį kaip ir atitinkamas įėjimas ar išėjimas.

Įrenginio būseną atitinkančių šviesos diodų būsenos pateiktos **2.2.2** lentelėje.

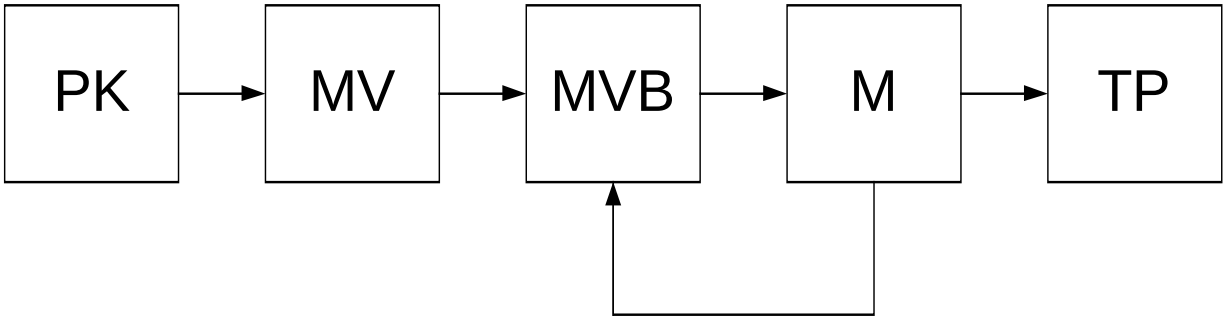
**2.2.2 lentelė. Indikatorių būsenos**

| PAVADINIMAS | BŪSENA |
|---|---|
| **Power ok** | Maitinimas įjungtas |
| **Ready** | Diagnostiniai tekstai praėjo sėkmingai ir valdiklis pasirengęs veikti |
| **Run** | Valdiklis vykdo vartotojo programą |
| **Battery low** | Išsikrovęs vidinis akumuliatorius |
| **I/Q exp.link** | Leidžiamas ryšys tarp tinklo valdiklių |
| **Com 1** | Perduodami duomenys iš pagrindinio kompiuterio |

Duomenims perduoti naudojami du valdiklio prievadai: comml - ryšiui su pagrindiniu kompiuteriu ir I/Q exp link - ryšiui su kitais valdikliais. Ryšiui su technologiniais įrenginiais numatytos įėjimų ir išėjimų grupės. Kiekvieno įėjimo ar išėjimo būseną žymi atitinkamas šviesos diodas, esantis viršutinėje valdiklio plokštėje.

## 2.3 Roboto ir mikrovaldiklio suderinimas

Manipuliatoriaus valdymo mikrovaldikliu sistemą galima pavaizduoti struktūrine schema, pateikta **2.3.1 pav**.:




**2.3.1 pav. Struktūrinė sistemos schema**

Čia: PK - pagrindinis kompiuteris; MV - mikrovaldiklis; MVB - manipuliatoriaus valdymo blokas; M - manipuliatorius; TP - technologinis procesas.

Pagal šią schemą programa, sudaryta pagrindiniame kompiuteryje, perduodama į valdančiąją ESM - mikrovaldiklį, kuris formuoja valdymo signalus galios keitikliui - manipuliatoriaus valdymo blokui. Manipuliatoriaus valdymo blokas pagal gautus valdymo signalus prijungia arba atjungia manipuliatoriaus variklių maitinimo įtampą. Taip pat šiame bloke registruojami variklių posūkio jutiklių ir galinių išjungiklių signalai.

Reikia suderinti tarpusavyje mikrovaldiklio ir manipuliatoriaus valdymo bloko informacinių signalų lygius ir sudaryti valdymo programą. Problemos sprendimą apsunkina tai, kad mikrovaldiklio išėjimai pritaikyti perjungti aukštas (iki 23 OV) įtampas, o manipuliatoriaus valdymo bloko įėjime naudojamos TTL lygių įtampos. Optimaliausias sprendimas būtų prijungti mikrovaldiklį prie MVB, išskiriant mikrovaldiklio jėgos dalį.

Varikliams valdyti informacinio lygio signalais galima pasinaudoti MVB numatyta rankinio valdymo galimybe.

Valdant rankiniu būdu, nuspaudus vieną ar kelis rankinio valdymo klavišus, sujungiami kontaktai, per kuriuos loginiai signalai siunčiami į valdymo schemą ir įjungiami atitinkami varikliai. Lygiagrečiai šiems klavišams prijungus mikrovaldiklio valdomus perjungimo elementus, sistemą galima valdyti programa. Tam galima panaudoti elektromagnetines reles, valdomas MV aukštos įtampos elementais. Dėl žymių trūkumų (dideli gabaritai, perjungimo triukšmas, aukštos valdymo įtampos) šio sprendimo taip pat atsisakome.

Mikrovaldiklio schemą sąlygiškai galima suskirstyti į tris dalis: loginę, tarpinę ir jėgos. Loginėje dalyje apdorojama informacija ir generuojami valdymo signalai, kurie tarpinėje dalyje sustiprinami iki jėgos dalies elementų komutavimo įtampų dydžio. Tarpinėje dalyje optronai galvaniškai atskiria loginę ir jėgos dalį. Jėgos dalis atlieka tiesioginį jo objekto elektros grandinių komutavimą. Schema veikia taip: loginėje dalyje suformuotas signalas siunčiamas į optrono šviesos diodo grandinę, kuri šviesos signalu įjungia fotosimistorių. Kadangi fotosimistorius įjungtas į jėgos dalies simistoriaus valdymo grandinę, tai įsijungia ir jėgos simistorius. Perjungiant reles, valdymo signalas siunčiamas į tranzistoriaus, kurio kolektoriuje įjungta relės ritė, bazę. Tranzistorius atsidaro, per ritę prateka srovė ir relės kontaktai susijungia.

Aišku, kad tiesioginis loginės dalies generuojamų signalų panaudojimas MVB valdyti yra komplikuotas, nes simistorių komutavimui naudojami impulsiniai signalai, o MVB valdyti reikalingi pastovūs aukšto arba žemo lygio signalai. Tokius lygius galima gauti programiniu būdu, tačiau tam būtų reikalinga sudėtingesnė programa, kuri užimtų daugiau valdiklio atminties. Vis dėlto panaudojus papildomą trigerių schemą, tokį valdymą galima realizuoti, ir šis būdas daug patikimesnis ir praktiškesnis negu ankščiau nagrinėtas, tačiau tam taip pat reikia montuoti papildomą schemą su 8 trigeriais, neskaitant kitų elementų.

Paprasčiausias ir patikimiausias sprendimas komutacijai panaudoti tarpinės dalies optrono fotosimistorių, kuris, kaip nustatyta iš eksperimentų, atsidaro nusiuntus valdymo signalą, kai prie jo išvadų prijungta +5V įtampa. Tačiau, kaip žinome, komutuojant pastovią įtampą, simistorius, išjungus valdymo signalą, savaime neužsidaro. Norint uždaryti reikiamą simistorių, nuo jo išvadų elektromagnetine rele trumpam atjungiama komutuojamoji įtampa, kartu atjungiant ir valdymo signalą.

Kadangi visiems simistoriams išjungti naudojama ta pati relė, tai išjungiant vieną simistorių komutuojama įtampa atjungiama visiems, tačiau atjungimo trukmė tokia trumpa, kad sistemos darbui tai neturi jokios įtakos.

Šio būdo privalumai : nereikia montuoti papildomos schemos, paprastas realizavimas, aukštas patikimumas ir didelė sparta. Taigi aišku, kad jį tikslinga naudoti perduodant informacinius signalus iš mikrovaldiklio į manipuliatoriaus valdymo bloką.

## 2.4. Paieškos optimizavimo uždaviniai

Paieškos tikslu gali būti tam tikro techninio ekonominio valdymo rodiklio pagerinimas, optimalaus sprendimo sudarymas projektavimo metu, optimalaus technologinio proceso režimo suderinimas ir t.t.

Formuluotėje visuomet yra tikslo funkcija

$$Q = Q(x, z), \qquad \textbf{(2.4.1)}$$

Išreiškianti optimizavimo rodiklio priklausomybę nuo valdomų kintamųjų vektoriaus $x = (x_1,..., x_k)$ ir vektoriaus atsitiktinių dydžių $z = (z_1,..., z_e)$. Kai kada optimizavimo uždaviniuose reikia apskaičiuoti kitus objekto rodiklius $h_j = h_j(x, z),\ \ j = 1,..., p.$

Valdomi kintamieji $x_i$ yra nepriklausomai vienas nuo kito ir optimizavimo metu gali būti keičiami tam tikrose ribose:

$$x_{i-} \le x_i \le x_{i+},\ \ i = 1,..., k; \qquad \textbf{(2.4.2)}$$

čia $x_{i-}$ ir $x_{i+}$ - ribinės įėjimo kintamųjų $x_i$ vertės, kurios parenkamos atsižvelgiant į objekto įrangos galimybes, saugumo technikos arba technologijos reikalavimus ir kt.

Priklausomu kintamuoju Q išrenkamas tas objekto funkcionavimo rodiklis (našumas, produkcijos savikaina, naudingumo koeficientas ir kt.), kurį būtina ir galima parenkant valdomuosius kintamuosius ribose **(2.4.2)**.

Atsitiktiniai kintamieji $z_i$ (kai kurie iš jų gali būti kontroliuojami) sudaro paieškos neapibrėžtumą ir mažiau ar daugiau pasireiškia optimizuojant gamybos technologinius procesus. Tai yra gamybos sąlygų sezoniniai pasikeitimai, žaliavos kokybinės sudėties parametrai, įrenginių dilimas, matavimo paklaidos, neužskaityti valdomi kintamieji ir t.t.

Atsižvelgiant į statistinių savybių kintamuosius $z_i$, galimybių jiems kontroliuoti formuluojamas paieškos tikslas ir parenkami optimizavimo metodai.

Paieškos tikslas formuluojamas kaip tam tikrų sąlygų sistema. Gali būti įvairūs paieškos optimizavimo uždaviniai. Pagrindiniai iš jų yra tokios formuluotės.

**1 uždavinys.** Jeigu parametrai $z_i$ kinta lėtai palyginti su laiko intervalu, reikalingu paieškai realizuoti, arba $z_i$ - pastovūs dydžiai, tai rodiklis Q beveik visai nepriklauso nuo kintamųjų $z_i$. Tardami, kad

$$Q(x, z) = Q(x),$$

paprastai formuluojame uždavinį taip. Surasti vektorių x*, kuris suteikia tikslo funkcijai

$$Q = Q(x) \qquad \textbf{(2.4.3)}$$

ekstremumą, atsižvelgiant į **(2.4.2)** apribojimus. Trumpiau šis uždavinys užrašomas:

$$Q(x) \to extr; \qquad \textbf{(2.4.4)}$$
$$x \in \Omega_x$$

čia $\Omega_x$ - paieškos sritis pagal nelygybių sistemą **(2.4.2)**. Jeigu funkcija Q(x,z) pasirinkta ir kintamieji $z_i$ paieškos metu matuojami, tai optimizavimo uždavinys gali būti sprendžiamas dėl kiekvienos realizacijos parametrų $z_i$, taikant išraišką **(2.4.4)**.

Kriterijus **(2.4.4)** dažnai taikomas projektavimo ir konstravimo uždaviniams, derinant naujus technologinius procesus.

**2 uždavinys.** Jeigu kintamieji $z_i$ - atsitiktiniai dydžiai su žinomu paskirstymo tankiu f(z) ir pasirinkta funkcija Q(x,z), tai galima surasti matematinę viltį (vidurkį) šios funkcijos

$$J(x) = M_z\{Q(x, z)\} = \int_{z \in \Omega_z} Q(x, z)\, f(z) dz,$$

čia $\Omega_z$ - sritis, kuriai priklauso kintamieji $z_i$.

Priklausomybė J(x) kai kada panaudojama kaip tikslo funkcija. Analogiškai **(2.4.4)** formuluojamas paieškos tikslas

$$J(x) \to \underset{x \in \Omega_x}{extr}. \qquad \textbf{(2.4.5)}$$

Optimizavimas tikslo funkcijos vidurkio **(2.4.5)**, palyginti su **(2.4.4)**, turi mažesnį efektą todėl, kad gautas paieškos proceso pastovus vektorius x = x* ne visuomet dėl kiekvienos realizacijos $z_i$ suteikia funkcijai Q(x,z) ekstremumą.

**3 uždavinys.** Dėl parametrų $z_i$ kitimo, laikui bėgant, optimalus taškas x* keičia savo koordinates. Šiuo atveju dreifas, kurio charakteristika apibrėžta priklausomybe

$$x^* = x^*(t).$$

Tokioje situacijoje natūralu objekto optimizavimo metu siekti funkcijos ekstremumo dėl kiekvieno laiko momento, t.y.

$$Q[x, z(t)] \to \underset{x(t) \in \Omega_x}{extr} \qquad 0 \le t \le T. \qquad \textbf{(2.4.6)}$$

Uždavinio sudarymas **(2.4.6)** priklauso ekstremaliam valdymui ir yra ekvivalentinis vektorinės funkcijos x(t) = x*(t) nustatymui, kuri suteikia ekstremumą funkcionalui

$$J_T = \frac{1}{T}\int_0^T Q[x(t), z(t)]dt. \qquad \textbf{(2.4.7)}$$

**(2.4.6)** ir **(2.4.7)** kriterijai taikomi, kai objekto savybių pasikeitimo sąlygose reikia pagerinti tam tikrą jo darbo rodiklį.

**(2.4.4)** ir **(2.4.6)** kriterijai paprasčiausi tipiški paieškos optimizavimo uždaviniuose. Tačiau praktika kelia įvairus paieškos tikslus, todėl yra daug uždavinių užrašymo formų. Pavyzdžiui, pasitaiko paieškos uždavinių su įvairiais funkciniais apribojimais, daugiakontūriai uždaviniai. Trumpai išnagrinėsime šių uždavinių ypatybes.

**Paieškos optimizavimo uždaviniai su funkciniais apribojimais.** Tarkim, technologinio proceso efektyvumas įvertinamas rodikliais: našumu N, savikaina S ir produkcijos kokybe K. Tada, atsižvelgiant į paminėtus rodiklius, paieškos tikslą galima formuluoti: surasti valdomųjų kintamųjų vektorių x*, kuris suteikia mažiausią vertę produkcijos savikainai

$$S = S(x), \qquad \textbf{(2.4.8)}$$

laikantis apribojimų

$$N(x) = N_0, \;\; K(x) = K_0; \qquad \textbf{(2.4.9)}$$

$$x_{i-} \le x \le x_{i+}, \;\; i = 1,...,k; \qquad \textbf{(2.4.10)}$$

čia $N_0$ ir $K_0$ - pasirinktos našumo ir kokybės rodiklio vertės.

Jeigu proceso rodikliai priklauso nuo atsitiktinių dydžių $z_i$, paieškos uždavinio sąlygos užrašomos analogiškai **(2.4.8)** – **(2.4.10)**, tačiau vietoje rodiklių S, N ir K vartojami jų vidurkiai.

Uždavinio su funkciniais apribojimais bendroji forma užrašoma taip:

$$Q(x) \to \underset{x(t) \in \Omega_x I \Theta}{extr} \qquad \textbf{(2.4.11)}$$

$$\Theta = \{h_j(x) \ge 0, \; j = 1,..., p\}. \qquad \textbf{(2.4.12)}$$

Uždaviniai, kurių formuluotė **(2.4.11)**, **(2.4.12)**, dažnai sprendžiami konstravimo ir projektavimo metu.

**Daugiakontūriai paieškos uždaviniai.** Gerai būtų surasti technologinio objekto režimą, kuris garantuotų didžiausią našumą, kokybės rodiklį ir mažiausią savikainą . Tačiau tokio režimo paprastai nėra ir jo suradimo uždavinio formuluotė greičiausiai bus nekorektiška. Iš tikrųjų daugumai technologinių objektų per didelis proceso forsavimas, norint padidinti našumą nepagrįstai padidina energijos ir žaliavos ekonomiją, todėl padidėja savikaina, be to, blogėja ir produkcijos kokybė.

Paieškos tikslo parinkimas ir jo formalizavimas – svarbi ir sudėtinga problema. Nuo jos tinkamo sprendimo daug priklauso optimizavimo sėkmė. Ne visuomet šią problemą galima patenkinamai išspręsti taikant kriterijus **(2.4.4)**, **(2.4.6)** ir **(2.4.11)**,**(2.4.12)**, kuriuose numatoma tik vieno rodiklio optimizavimas. Praktikoje kai kada būtina pagerinti kelius objekto funkcionavimo rodiklius ir tam tikslui formuluojami ir sprendžiami daugiakriteriai paieškos uždaviniai, taikant vektorinį kriterijų.

Tarkim, parinkta tam tikra vektorinė tikslo funkcija

$$Q(x) = [Q_1(x),...,Q_M(x)]. \qquad \textbf{(2.4.13)}$$

Atskirų tikslo funkcijų ekstremumai pasiekiami optimaliuose taškuose

$$x_1*,...,x_M*. \qquad \textbf{(2.4.14)}$$

Jeigu visi vektoriai (2.14) lygūs

$$x_i* = x_j*, i \neq j, i = 1,...,M, \qquad \textbf{(2.4.15)}$$

tai **(2.4.15)** yra daugiakriterio uždavinio sprendimas.

Tačiau **(2.4.15)** lygybės beveik niekuomet praktikoje nebūna. Todėl formuluojamas uždavinys rasti tam tikrą kompromisinį sprendimą. Ieškant funkcijų $Q_j(x)$ mažiausios vertes , kompromisinis sprendimas $x_* \in \Omega_x$ bus efektyvus taškas, jeigu jį atitinka nelygybės:

$$Q_j(x_*) \leq Q_j(x), \quad j = 1,...,M,$$

be to, viena iš jų turi būti griežta. Iš to apibrėžimo išplaukia, kad yra aibė $\Omega_*$ efektyvių taškų. Ji vadinama kompromiso aibė.

Vienas optimalus sprendimas parenkamas iš kompromiso aibės, atsižvelgiant į atskirų kriterijų svarbumą. Todėl, formuluojant daugiakriterį paieškos uždavinį, reikia atlikti tam tikrą vektorinio kriterijaus sujungimą. Tam tikslui taikomi įvairūs sujungimo metodai: sujungimas kokybiniu atžvilgiu lyginamųjų ir ranginių kriterijų, globalinio kriterijaus sudarymas ir kt.

Dažnai taikomi sujungimo metodai, numatantys tam tikros skaliarinės funkcijos

$$Q(x) = \Phi[Q_1(x),...,Q_M(x)], \qquad \textbf{(2.4.16)}$$

kurioje kiekviena atskira tikslo funkcija užima atitinkamą vietą pagal savo svarbą ar naudą, sudarymą. Vienintelis vektorius $x^* \in \Omega_x$, kuris suteikia **(2.4.16)** funkcijai ekstremumą, yra daugiakriterio uždavinio optimalusis sprendimas. Pavyzdžiui, dėl palyginimų kokybiniu atžvilgiu $Q_j(x)$, tokia tikslo funkcija gali būti suma

$$Q(x) = \sum_{j=1}^{M} \psi_j Q_j(x),$$

čia $$\psi \geq 0, \qquad \sum_{j=1}^{M} \psi_j = 1.$$

Svorio koeficientai $\psi_j$, kurie nurodo j-jo kriterijaus svarbą (svorį), gali būti išrinkti taikant eksperimentinius įverčius, atsižvelgiant į fizikines uždavinio savybes ir t.t.

Kai kada pavyksta $Q_j(x)$ sujungti į vieną skaliarinę funkciją, išreiškiančią techninį ekonominį rodiklį. Tada daugiakriterio uždavinio formuluotė pagrįstesnė ir natūralesnė. Paprastas tokio atskirų kriterijų sujungimo pavyzdys gali būti technologinio proceso automatizacijos uždavinyje, kurio efektyvumas kaip **(2.4.8)** – **(2.4.10)** įvertinamas našumu N, savikaina S, produkcijos kokybe K. Formuluojant tikslo funkciją, šiame uždavinyje galima panaudoti skaliarinę funkciją, išreiškiančią pelną per laiko vienetą

$$\Pi(x) = \{L[K(x)] - S(x)\}N(x), \qquad \textbf{(2.4.17)}$$

čia L – produkcijos vieneto kaina, priklausanti nuo kokybės K. Nesunku sudaryti atskirus išraiškos **(2.4.17)** atvejus. Tai gali būti pelnas, esant tam tikrai pastoviai našumo arba kokio nors kito rodiklio vertei **[5]**.

# 3. ROBOTŲ VALDYMO OPTIMIZAVIMO ALGORITMAI

## 3.1 Automatinio judesio planavimo ir programavimo uždaviniai

Iki šiol pramoninių robotų judesius programavo žmogus – operatorius, kuris mokymo režimu formavo reikiamą judėjimo programą ir įrašinėjo ją į atmintinės modulį. Tokio metodo trūkumai – didelis darbo našumas, ilgas apmokymo procesas (reikalaujantis operatoriaus aukštos kvalifikacijos) ir judėjimo programos nustatytas parašymo algoritmas. Todėl negalima operatyviai keisti programos realioje aplinkoje, kuri gali labai skirtis nuo idealios. Aišku, kad robotams, veikiantiems neapibrėžtoje ir ekstremalioje aplinkoje, kuri pavojinga žmonių gyvybei, toks programavimo metodas tampa neefektyvus. Praktiškai jis neturi didelio tikslumo ir gali sukelti avariją.

Tai skatina naujų automatinio programavimo algoritmų kūrimą, kurie užtikrintų roboto prisitaikymą prie aplinkos. Toks požiūris į problemą reikalauja įvertinti papildomą informaciją, kuri gaunama eksploatuojant robotą, ir tam tikru būdu automatiškai koreguoti judėjimo programą.

Pereisime prie tikslios algoritminės sintezės ir programinių judesių(PJ) optimizavimo uždavinio formuluotės. Pramoninių robotų manipuliatorių dinamiką išreiškia diferencialinė lygtis:

$$A(q,\xi)\,\ddot{q} + b(q,\dot{q},\xi) = u, \quad t \in [t_0, t_T]. \tag{3.1}$$

Čia q = q (t) – laiko momentų apibendrintų koordinačių n-tasis vektorius; $\dot{q},\ddot{q}$ - greitis ir pagreitis, u – valdomų jėgų arba momentų, kuriuos sukuria valdomoji pavara, n-tasis vektorius; $\xi$ - parametrų n-tasis vektorius (inercinės grandžių charakteristikos, trinties koeficientai ir t.t.); $t - t_0 = T$ - judėjimo laikas; $A(q,\xi), b(q,\dot{q},\xi)$ - pasirinkti matrica-funkcija ir vektorius-funkcija matmenys *n x n* ir *n*. Manipuliatoriaus judesiai ir pavarų valdymas praktiškai labai riboti:

$$q(t) \in Q_q, \quad \dot{q}(t) \in Q_{\dot{q}} \text{ prie visu } t \in [t_0, t_T]; \tag{3.2}$$

$$u(t) \in Q_u \text{ prie visu } t \in [t_0, t_T]; \tag{3.3}$$

čia $Q_q$, $Q_{\dot{q}}$, $Q_u$ - daugybės, kurias parenka roboto konstrukcija.

Manipuliatoriaus būseną laiko momentu t nustato vektorius

$$x(t) = \begin{vmatrix} q(t) \\ \dot{q}(t) \end{vmatrix}.$$

Uždavinys, aprašantis judėjimą, yra toks, manipuliatorius, judantis iš pradinio pasirinkto taško $x_0$ į norimą galutinį tašką $x_1$, turi atitikti sąlygas:

$$q(t_0) = q_{0,} \quad q(t_T) = q_1;$$
$$\dot{q}(t_0) = \alpha_{0,} \quad \dot{q}(t_T) = \alpha_{1.} \qquad \textbf{(3.4)}$$

Dabar galime tiksliai suformuluoti PJ apibrėžimą. PJ $q_p(t)$ vadinsime dinaminės lygties **(3.1)** sprendimu, kuris atitinka apribojimus **(3.2)**, **(3.3)** ir **(3.4)**. PJ kokybei įvertinti pasirenkamas vienas ar keli optimizavimo kriterijai. Tokiu kriterijumi gali būti, pavyzdžiui, energetinių išlaidų mažinimas arba judėjimo laiko mažinimas. Kai PJ $q_p^0(t)$ funkcionalas turi mažiausią vertę, vadinsime optimaliu. Pažymėsime, kad PJ $q_p^0(t)$ minimizacijos judėjimo laikas T užtikrina didžiausią roboto naudingumo koeficientą.

Žinant optimalų PJ $q_p^0(t)$ ir įrašant jį į dinaminę lygtį **(3.1)**, galima gauti toki programinio judesio optimalų dėsnį:

$$u_p^0(t) = A[q_p^0(t), \xi]\ddot{q}_p^0(t) + b[\dot{q}_p^0(t), \dot{q}_p^0(t), \dot{q}_p^0(t), \xi] \quad . \qquad \textbf{(3.5)}$$

Tokiu būdu roboto optimalaus valdymo uždavinys tiesiogiai susietas su jo PJ optimizavimu.

Sprendžiant PJ optimizavimo uždavinius, tinka klasikiniai variacinio skaičiavimo metodai. Tačiau įvertinant apribojimus **(3.2)**,**(3.3)** ( būtinumas vengti kliūtis), tokie metodai tampa neefektyvūs arba apskritai netinkami. Todėl atsiranda poreikis kurti specialus metodus, kuriuose bus įvertinta robotų PJ optimizavimo uždavinio specifika. Bendras koordinates $q_i$ atstoja kampai tarp grandžių (porom sukant) arba grandžių ilgiai (porom žingsniuojant). Todėl manipuliatorius, turintis *l* grandžių, turi $n \le 3l$ laisvo judėjimo laipsnių.

Pagal manipuliatoriaus konfigūraciją q(t) vienareikšmiškai apibūdinama jo griebtuvo padėtis $r(t) = |\, r_i(t) \,|^3_{i=1}$ (kartais į vektoriaus r komponentų skaičių įtraukiami ne tik pasirinkto užgriebimo taško dekadinės koordinatės, bet ir nukreipiantys (kosinusai). Todėl pasirenkamas operatorius Φ:

$$\Phi[q(t)] = r(t), \quad t \in (t_0, t_T), \tag{3.6}$$

kuris aprašo manipuliatoriaus kinematiką.

Išraiška D = $\Phi(Q_q)$ yra manipuliatoriaus pasiekimo riba. M(q) pažymėsime išraiška $D \subset R^3$, kuri užima manipuliatorius konfigūracijoje q. Tegul $P \subset R^3$ yra objektai, kurie sudaro kliūtis. Įvesime kliūčių išraiškas

$$\sigma(q) = \begin{Bmatrix} 1, \; jeigu P \cap M(q) \neq \emptyset; \\ 0 - kitu\_atveju \end{Bmatrix}, \tag{3.7}$$

kurios informuoja, ar trukdo manipuliatoriui kliūtis P konfigūracijoje q.

Griebtuvo perkėlimo į užsibrėžtą tašką uždavinys formuluojamas taip. Tegul D srityje pasirinktas taškas $r_*$ ir tegul $q(t_0) \in Q_q, \dot{q}(t_0) \in Q_{\dot{q}}, \sigma[q(t_0)] = 0$ . Reikia sukurti programinį judesį $q_p(t)$ tokį, kad būtu įvertinti apribojimo sąlygos

$$\begin{aligned} &q_p(t_0) = q_0, \Phi[q_p(t_T)] = r_*; \\ &\dot{q}(t_0) = \alpha_0, \dot{q}_p(t_T) = 0 \end{aligned} \tag{3.8}$$

ir kliūties apėjimo sąlygos

$\sigma[q_p(t)] = 0$ prie visu $t \in [t_0, t_T]$. **(3.9)**

Uždavinys stebiant griebtuvu pasirinktą trajektoriją atrodo kitaip. Pagal pasirinktą griebtuvo trajektoriją $r_*(t)$ ir manipuliatoriaus pradinę būseną $x_p(t)$ reikia sudaryti PJ $q_p(t)$ tokį, kad

$\Phi[q_p(t)] = r_*(t)$ esant visiems $t \in [t_0, t_T]$ **(3.10)**

ir būtu įvykdyti kliūties apėjimo reikalavimai **(3.9)**.

Konstrukciniai apribojimai **(3.2) - (3.3)** apibendrintoms koordinatėms ir išorinės kliūtys žymiai pasunkina suformuluotų uždavinių sprendimą. Ne kiekvienai griebtuvo trajektorijai $r_*(t)$, esančiai pasiekimo srityje D, egzistuoja ją atitinkantis nepertraukiamas PJ $q_p(t)$. Kliūčių egzistavimas priverčia atmesti daug linijinės lygties **(3.6)** sprendimų kaip netinkamus. Gali iškilti nesklandumų faktiškai sudarant PJ. Galima pasakyti, kad įvertinimas realių apribojimų ir kliūčių atliekant manipuliatoriaus PJ algoritminę sintezę, sukuria įvairaus pobūdžio be išeitines situacijas. Atradimas ir apėjimas tokio pobūdžio situacijų reikalauja specialios analizės ir pradinio PJ planavimo. Dabar žinomi įvairūs būdai sprendžiant manipuliatoriaus PJ planavimą.

PJ planavimas pasižymi geromis savybėmis:

- pasiekimo srities D (arba konfigūracijos daugybės $Q_q$,) daugiakampių apibrėžimas, norint sudaryti grafu planą;
- prisitaikymas prie kliūčių grafo plane;
- PJ pagal jo planą karkaso sudarymas;

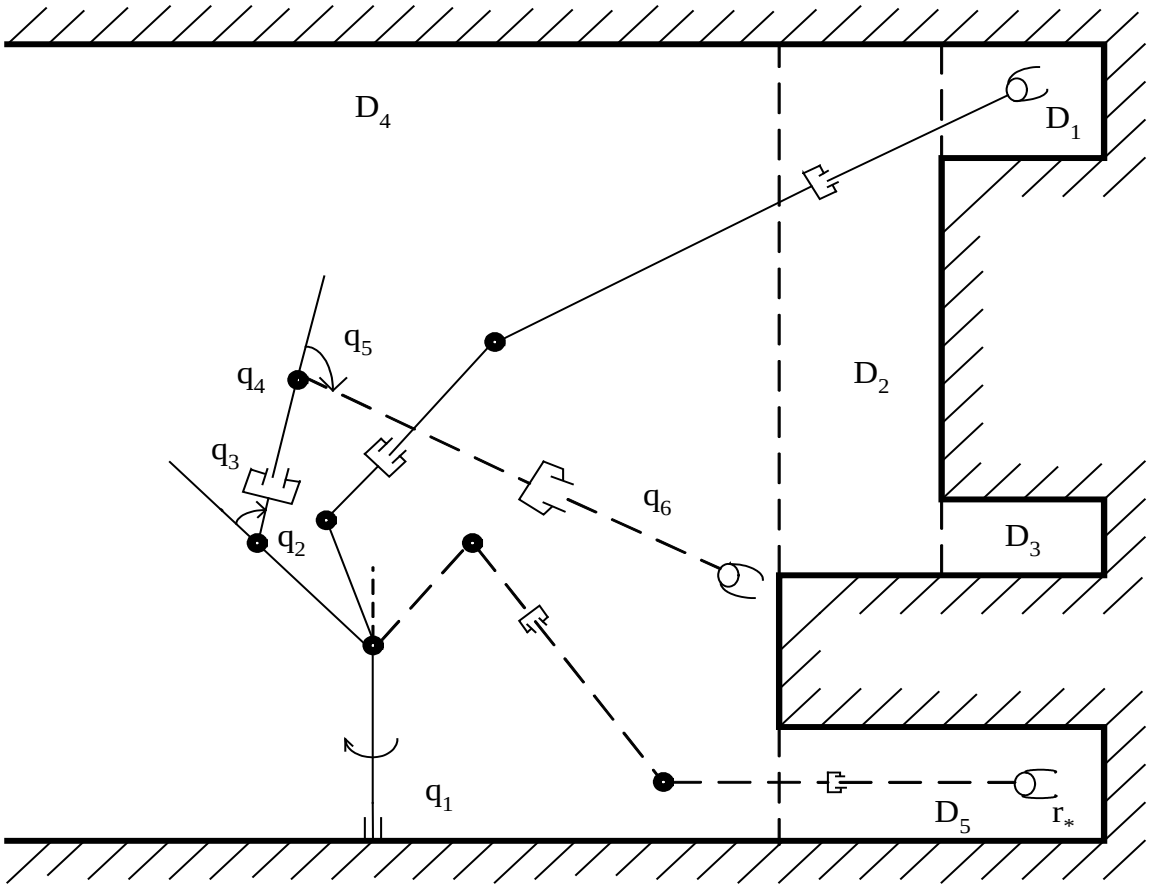
Siūlomo PJ planavimo metodo pagrindinis privalumas yra tas, kad be išeitinių situacijų apėjimo uždavinius galima suvest prie palaipsnių vietinių uždavinių judėjimo programavimo. Sprendžiant minėtus uždavinius, galima naudotis žinomais PJ sudarymo algoritmais. Šitie algoritmai yra vietiniai galvojant apie tai, kad be papildomo PJ planavimo jie patys gali sudaryti (o faktiškai ir sudaro) situacijas be išeities netgi ir tada, kai praktiškai PJ egzistuoja. Todėl pirminis planavimas gali realiai praplatinti priimtumo sritį kaip jau žinomų ir kaip vietinių PJ planavimo algoritmų, kurie bus siūlomi ateityje.

Reikia pažymėti, kad kitas nagrinėjamas adaptyvus sintezės ir PJ optimizavimo metodas, pagrįstas specialia PJ parametrizacija, su **(3.4)** apribojimu įvertinimu, nereikalauja pradinio planavimo. Šis metodas yra globalus. Bet esant sudėtingoms kliūtims, apribojimų sprendimo sritis, kuria sudaro PJ adaptyvus sintezės metodas, gali turėti daug ryšių. Tokiais atvejais toks metodas irgi reikalauja pradinio PJ planavimo, kas padeda lokalizuoti vieną ar kitą nelygumų-apribojimų sprendimų sritį.

## 3.2 Planavimo metodologija ir optimizavimas grafu plane

Susipažinsime plačiau su metodologiniais programavimo aspektais ir robotų PJ optimizavimu, kai yra kliūtys. Pirmiausiai panagrinėsime roboto griebtuvo nustatymą ties pasirinktu tašku. Šitą uždavinį geriausiai spręsti keturiais etapais.

Pirmuoju etapu daugiakampiu $D_1$, …, $D_v$ apibrėžiama darbinė sritis. Tokio apibrėžimo pavyzdys parodytas paveiksle **3.1** .

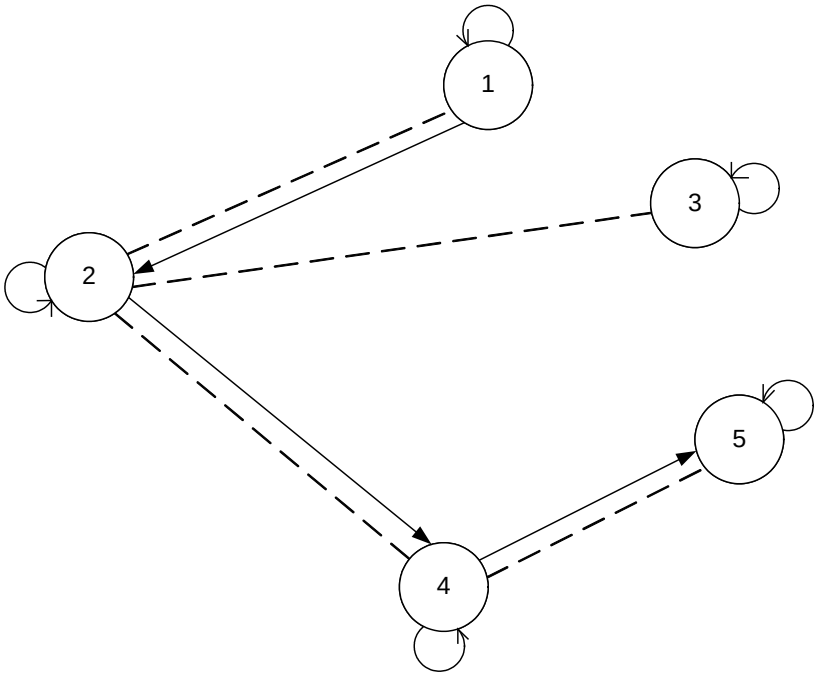



**3.1 pav**. Manipuliatorius kliūčių aplinkoje

Geometrinių modelių darbinės aplinkos $D_1$, …, $D_i$ pavidalo kuriamas atitinkamas grafas $G_D$, kurio viršūnė su indeksu $i$ atitinka $D_i$, o briaunos sujungia viršūnes $i$ ir $j$ tais atvejais kai daugiakampiai $D_i$ ir $D_j$ susikerta. Tokiu principu sudarytas grafas $G_D$ aprašo darbinės zonos

struktūrą ir yra PJ planavimo pagrindas. Manipuliatoriui, parodytam **3.1** paveiksle, atitinka grafas, pavaizduotas **3.2** paveiksle.

Antras etapas – kelio pasirinkimas grafe $G_D$. Kad būtų aiškiau, tegul $r(t_0) \in D_{i0}$ , o $r_* \in D_{i*}$. Jeigu pasirodys, kad $i_0 = i_*$ , tai planavimo nereikia ir galima iš karto sudaryti PJ pagal vieną ar kitą $D_{i_n},...,D_{i_n}$, vietinį algoritmą. Jeigu $i_0 \neq i_*$ , tada kelią $i_0, i_1,..., i_n = i_*$ , sujungiantį $G_D$ grafe viršūne $i_0$ su viršūnę $i_*$ , pavadinsime griebtuvo judėjimo planu. Tokiu būdu planas sąlygoja daugiakampių, per kurios reikia vedžioti griebtuvą į numatytą tašką $r_*$, eiliškumą. **3.2** paveiksle, PJ planas, grafų plane $G_D$ pažymėtas rodyklėmis, kelias 1, 2, 3, 4, 5.




**3.2 pav**. Grafų planas ir PJ planas

Trečiuoju etapu parinkus planą sudaromas PJ karkasas, tai yra tokių konfigūracijų eiliškumas $q_0,...,q_n$, kad $\Phi(q_0) = r(t_0),...,\Phi(q_n) = r_*, (q_i) \in D_i$. Jeigu pereinant iš $D_{i_k}$ į $D_{i_{k+1}}$ atsirado be išeitinė situacija, tai iš grafų plano $G_D$ pašalinama atitinkama briauna ir naujame

grafe sudaromas naujas griebtuvo judėjimo planas. Tuo pačiu realizuojama savotiška adaptacija esant kliūtims.

Pagaliau ketvirtame etape pagal PJ karkasą sudaromas pats PJ, atsižvelgiant į visus apribojimus **(3.2)**, **(3.3)**. Šiam uždaviniui išspręsti naudojami pateikti žemiau parametrinės sintezės ir PJ optimizavimo algoritmai.

Aprašysime karkaso sudarymo vetiniai optimalų algoritmą. Tam įvesime į $Q_q$ n-ją grotelę H su žingsniu h. PJ karkasą galima sudaryti pagal grotelės H mazgus.Tarkime, kad dalis PJ karkaso $q_0,...,q_{k+1}$ jau sudaryta . Kita konfigūracija $q_{k+1}$ renkama iš artimiausių groteles H taškų $q_k$ su vietinio optimizavimo sąlygą.

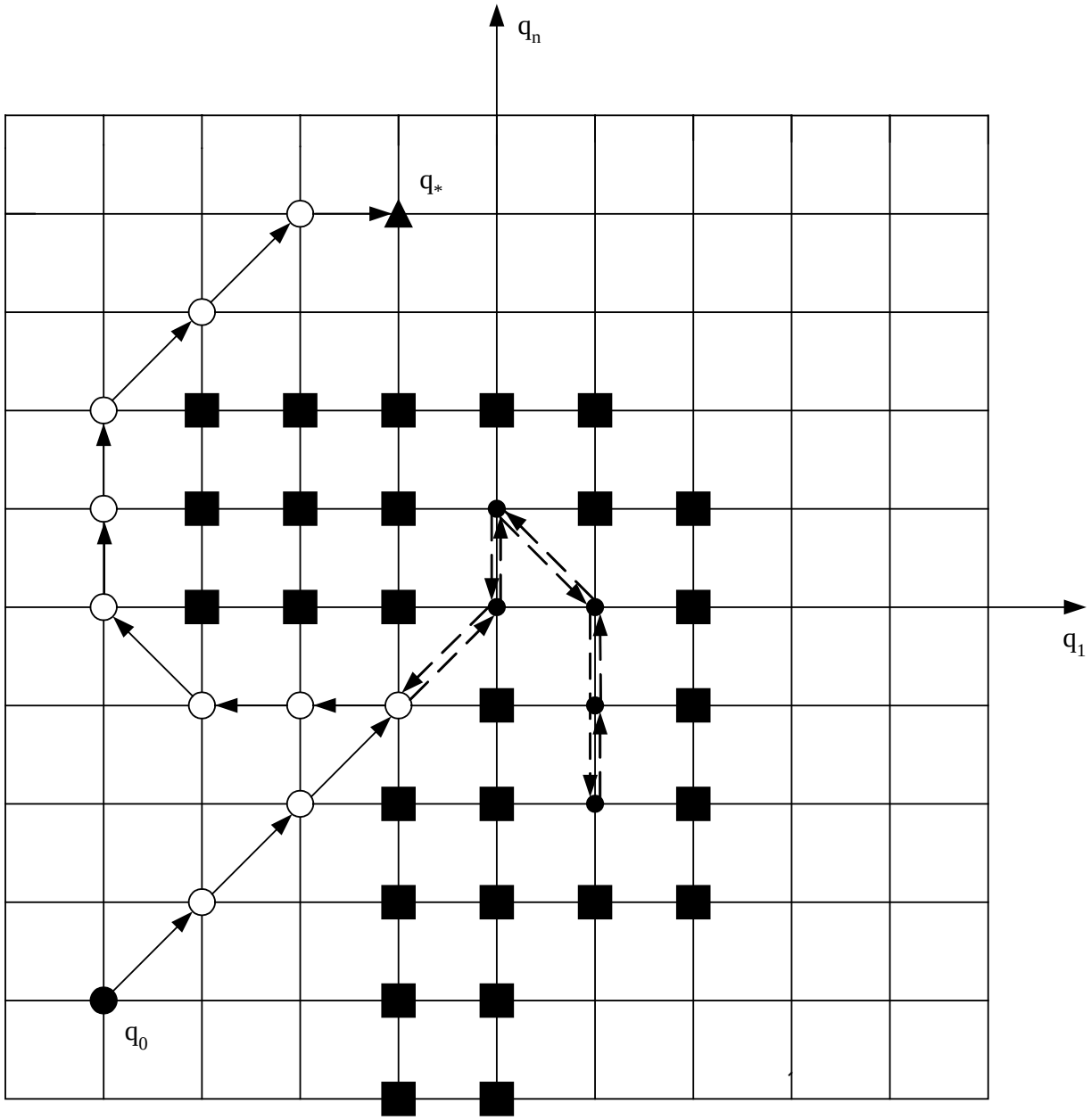
$$\left\|\Phi(q_{k+1}) - r_*\right\| = \min. \qquad \textbf{(3.11)}$$

Taip pat turi būti įvertinti apribojimai: 1) $q_{k+1} \in Q_q$; 2) $\sigma(q_{k+1}) = 0$; 3) $q_{k+1} \neq q_i, i = 0,...,k$. Galiausiai sudaromas karkasas be ciklų, kurio elementai $q_i$ atitinka konstrukcinį apribojimą **(3.2)** ir kliūčių apėjimo reikalavimus. Jeigu neegzistuoja artimiausios konfigūracijos su $q_k$ ir neatitinka apribojimai 1-3, tai paimama $q_{k+1} = q_k$, o numeris k sumažinamas vienetu. Algoritmas tęsia savo darbą iki tol, kol $\left\|\Phi(q_{k+1}) - r_*\right\|$ ne bus mažesnė negu ε. Jeigu žinoma numatyta konfigūracija $q_*$ tokia, kad $\Phi(q_*) = r_*$, tai vietoj kriterijaus **(3.11)** galime taikyti kriterijų

$$\left\|q_{k+1} - q_*\right\| = \min. \qquad \textbf{(3.12)}$$

Panagrinėsime vietiniai optimalaus algoritmą, kuris aprašo PJ karkasą. Konfigūracijų erdvėje sudarysime grotelę H su žingsniu h ir pažymėsime mazgus, kurie atitinka pradinę ir galutinę manipuliatoriaus padėtį. **3.3** paveiksle šitie mazgai pažymėti uždažytais apskritimais ir trikampiais, o mazgai, kurie atitinka konfigūracijas, kai manipuliatorius kerta kliūtis, pažymėti uždažytais kvadratais. Taikydami ankščiau minėta algoritmą su optimizavimo kriterijumi **(3.12)**,

gausime PJ karkasą, pažymėta **3.3** paveiksle nenudažytais apskritimais, jungiantį pradinę konfigūraciją $q_0$ su galutine $q_*$.




**3.3 pav**. PJ karkaso sudarymo vietiniai optimalus algoritmas

PJ karkaso sudarymo metu atsirandantį ciklą, kuris parodytas paveiksle **3.3,** brūkšniuota linija, galime atmesti kaip prieštaraujantį sąlygai 3.

Nagrinėtas PJ karkaso sudarymo algoritmas yra iš tikro paieškinis kelio algoritmas grafe H, tiksliau nežinomame pografe $H_p$ šito grafo mazguose q, kurio $\sigma(q) = 0$. Šito pografo iš anksto pasirinkti praktiškai neįmanoma, todėl kad kliūtys P dažniausiai nežinomos, o jeigu ir

žinomos, tai daugybės $Q_p$ sudarymas, kai $\Phi(p) = Q_p$, labai sunkus uždavinys. Todėl labai svarbu adaptuoti grafą H prie kliūčių, gaunant informacija apie susikirtimus su kliūtimis, atrenkant ir atmetant negalimus mazgus kaip sąlygą **(3.7).**

Panagrinėsime ryšius tarp planavimo ir PJ sudarymo. Tegul turime griebtuvo judėjimo planą $i_0,...,i_n$. Jam atitinka PJ karkasas, nulemiantis konfigūracijų $q_0,...,q_u$ eiliškumą, per kurias turi eiti PJ. Faktiškam PJ sudarymui pagal jo karkasą, atsižvelgiant į konstruktyvius apribojimus **(3.2),** galima panauduoti parametrinės optimizavimo metodus.

Panagrinėsime dabar PJ planavimo ypatumus uždavinyje, kur sekama pasirinkta griebtuvo trajektorija. Priminsime, PJ $q_p(t)$ turi atitikti sąlygas **(3.9)**, **(3.10)**. Bet kurią trajektoriją $r_*(t)$, kuri yra pasiekimo srityje D, galima sekti manipuliatoriaus griebtuvu tik tuo atveju , jeigu atvaizdas $\Phi : Q_q \to D$ yra atviras. Tokiu atveju PJ sudarymui galima nauduoti vietinius algoritmus.

Bet praktikoje atvaizdas Φ dažniausiai nėra atviras. Todėl pirmame etape siūlome apjungti daugybę $Q_q$ lygiagretainiais $Q_1,\ldots, Q_M$ taip, kad vietiniai atvaizdai $\Phi_i : Q_i \to D_i$, čia $D_i = \Phi(Q_i)$ , $\bigcup_{i=1}^{M} Q_i \equiv Q_q$ – būtų atviri. Panagrinėsime grafą $G_Q$, i-tasis viršutinis taškas atitinka $Q_i$, o kraštinės sujungia viršūnes su numeriais i ir j tik tais atvejais, jeigu $Q_i$ liečia $Q_j$ nemažiau kaip 3 matmenų ribas. Sudaryta tokiu būdu grafą $G_Q$ pavadinsime PJ planų grafu srityje $Q_q$. Prieš suteikiant viršūnei $G_Q$ i-jai ortą $e_i$ į $R^M$ (i-sios viršūnės kodas) kiekviename taške $r \in D$ parinksime M-indeksą:

$$I(r) = \sum_{\{i:\Phi(q)=,q\in Q_j\}} e_i \quad . \qquad \textbf{(3.13)}$$

Visi taškai r srityje D, turintis vienoda indeksą, pavadinsime zoną. Kiekvieną zoną užkoduosime jos indeksu **(3.13)**. Tuo pačiu pasiekimo sritį D dalinimą į galutinias skaičių zonas nesusikertančias pagal vidinius taškus. Tegul pasirinkta griebtuvo trajektorija $r_*(t)$. Galima priimti, kad $r_*(t) \in D$ prie visu $t \in [t_0, t_T]$ ir $r_*(t_0) = \Phi(q_0)$. O perėjimų skaičius, sekant

trajektoriją $r_*(t)$, iš vienos zonos į kitą galutinis. Apibrėžyme zonų $I_o,...,I_k$ eiliškumą, per kurias pereina $r_*(t)$. Pagal pradinę konfigūraciją $q_0$ rasime grafo $G_{i_0}$ viršūnės $e_{i_0}$ kodą , tokį, kad $q_0 \in Q_{i_0}$. Grafų plane $G_Q$ sudarysime kelią, prasidedantį viršūne su kodu $e_{i_0}$ ir atitinkantį reikalavimą

$$I_j O e_{i_j} = e_{i_j}, \quad j = 0, \ldots, k, \qquad \textbf{(3.14)}$$

čia O - pokoordinatinis vektorių daugybes simbolis. Tegul šis kelias pereina per grafų plano $G_Q$ viršūnes su kodais $e_{i_0},...,e_{i_K}$.

Eiliškumas $\{e_{i_j}\}_j^k = 0$ vadinamas PJ planu uždaviniuose, kur sekama manipuliatoriaus griebtuvu pasirinktą trajektoriją. Kiekvienai griebtuvo trajektorijai, bendru atveju, atitinka keli PJ planai. Dauguma tokiu planu charakterizuoja galimus trajektorijos sekimo būdus ir manipuliatoriaus manevravimus. Ne visi planai leidžiami ir vienodi pagal reikalavimą kliūtims. Atsižvelgiant į tai apibrėžyme optimalu PJ planą. Su kiekvieną grafų plano $G_Q$ viršūnę surišime matmenį

$$W(j)[\int \sigma(q)dq][\mu(Q_j)]^{-1} \quad , \qquad \textbf{(3.15)}$$

čia $\mu(Q_j)$ - tūris $O_j(\int \sigma(q)dq$ labai paprastai paskaičiuojamas, pavyzdžiui pagal Monte-Karlo metodą).

Planą $\{e_{i_j}\}_j^k = 0$ vadinsime optimaliuoju (suprasdami kad reikia vengti kliūtis), jeigu jis minimizuoja $maxW(i_j)$ . Šitą apibrėžimą atitinkantis PJ praeina per mažiausiai “apkrautus” kliūtimis lygiagretainius $Q_i$.

PJ planavimo esmė turint konstruktyvinius apribojimus ir kliūtis yra tokia, kad jis leidžia globalinius apėjimo uždavinius be išeities suvesti prie paprastų uždavinių, kuriu sprendimui galima nauduoti vietinius algoritmus. Jeigu pasirinktai trajektorijai planų grafe $G_Q$ neegzistuoja PJ plano, tai iš tuo galime spręsti, kad principingai negalime atmesti trajektoriją. Tarkime, kad PJ egzistuoja; tada iškyla klausimas : kaip faktiškai sudaryti PJ pagal pasirinktą planą?

Pirmiausia pažymėsime, kad santykis **(3.14)** garantuoja, kad kiekvienai daugybei $Q_{i_j}$ galima sudaryti atitinkamą PJ fragmentą naudojant vietinius algoritmus. Iš tuo kad, PJ planas $\{e_{i_j}\}_j^k = 0$ lemia kelią grafų plane $G_Q$, išplaukia, lygiagretainiai $Q_{i_n}$ ir $Q_{i_{n+1}}$, atitinkantis esantiems grieta plano elementams $i_n$ ir $i_{n+1}$, turi nemažiau kaip 3 matmenų grįžtamuosius ryšius. Tai reiškia, kad atsiras mažiausiai tris apibendrintos koordinatės, kurios užtikrina sekimą $r_*(t)$, kai atitinkamas PJ pereina iš $Q_{i_n}$ į $Q_{i_{n+1}}$. Bendrą PJ sudarymo pagal planą uždavinį, patogu spręsti dviems etapais. Pirmiausia reikia sudaryti PJ karkasą, atsižvelgiant į kliūčių apėjimo sąlygas **(3.9)**, o po to – patį PJ , atsižvelgiant į konstrukcinius apribojimus **(3.2)**.

Įvesime PJ karkaso apibrėžimą uždavinyje kur griebtuvų sekama pasirinkta trajektorija. Tegul trajektorija $r_*(t)$ yra išreikšta taškų nuoseklumų $r_0,...,r_k$ (tokia griebtuvo trajektorijos diskretinė forma dažnai pasitaiko praktikoje). Aprašysime zonų nuoseklumą $I_0,...,I_k$, per kurias praeina ši trajektorija. PJ karkasu vadinsime konfigūracijų nuoseklumą $q_0,...,q_k$, atitinkanti planui $\{e_{i_j}\}_j^k = 0$, $\Phi(q_j) = r_j$

ir

$$\left\| q_{j-1} - q_j \right\| \prec \delta, \quad j = 1,...,k, \qquad \textbf{(3.16)}$$

čia $\delta$ - pakankamai mažas skaičius.

Aprašysime bendra PJ karkaso sudarymo metodą pagal jo planą. Įvesime operatorių $A: Q_q \times D \to Q_q$ tokį, kad bet kuriems $q^0 \in Q_q$ ir $r \in D$ teisinga išraiška

$$\Phi[A(q^0, r)] = r\,.$$

Kadangi $A[q^0, \Phi(q^0)] = q^0$, tai operatorių A vadinsime klaidingai grįžtamuoju, lyginant su operatoriumi $\Phi$. Tegul būna duota diskretinė užgriebimo trajektorija $r_0, r_1,..., r_k$ tokia , kad matmenys $\left\| r_{j-1} - r_j \right\|$ būtu pakankamai maži. Todėl pradinė manipuliatoriaus konfigūracija

$q_0$ bus tokia, čia $\Phi(q_0) = r_0$, $q_0 \in Q_{i_0}$. Sekantis PJ karkaso elementai surasime pagal rekurentinę išraišką

$$q_{i+1} = A(q_j, r_{j+1}) \quad j = 1,...,k. \qquad \textbf{(3.17)}$$

Panagrinėsime rekurentinės išraiškos savybes **(3.17)**. Dėl to kad operatorius A yra klaidingai grįžtamasis algoritmo trajektorijoje **(3.17)** turime

$\Phi(q_j) = r_j$, j = 1, … , k.

Tai sako apie tai, kad išraiška **(3.17)** galima nauduoti sprendžiant atvirkštinės manipuliacijos uždavinius, tai yra sprendžiant kinematikos lygtis **(3.6)** atsižvelgiant į q ir pasirinktą r.

Nagrinėjamas metodas turi užtikrinti karkaso nenutraukimą atitinkamai su santykiais **(3.16)**. Toks reikalavimas kelia papildomus apribojimus operatoriui A. Algoritmo trajektorijoje **(3.17)** iš mažumos $\|r_{j-1} - r_j\|$ išplaukia mažuma $\|q_{j-1} - q_j\|$ visiems j = 1,…, k. Suformuota savybė vadinsime operatoriaus A nepertraukimu.

Dėl to, kad operatorius A klaidingai grįžtamasis pakankama sąlyga jo nepertraukiamumo yra funkcijos A nepertraukiamumas pagal antrą argumentą. Bet praktikoje dažniausiai A nėra nepertraukiama funkcija *r*, todėl kad operatoriaus A sritis labai didelė .

PJ karkasas sudaromas pagal palaipsnio perėjimo planą nuo vieno lygiagretainio $Q_{i_j}$ prie kito $Q_{i_{n+1}}$. Visi šitie lygiagretainiai turi savybę, kad atvaizdas $\Phi : Q_i \to D_i$, $D_i = \Phi(Q_i)$ yra atviras. Tuo pačiu užtikrinamas operatoriaus A nepertraukiamumas ir PJ karkaso sudarymo galimybė.

Pasiūlytas metodas yra universalus: kai vietoj operatoriaus A galima nauduoti operatorius, kurie aprašomi bet kurio algoritmo atvirkštinės manipuliacijos uždavinio sprendimu. Be to reikalavimai operatoriui A, klaidingam panašumui ir nepertraukimui, yra būtinos sąlygos PJ karkaso egzistavimui, o taip ir PJ. PJ sudarymas pagal jo karkasą su konstruktyviniais

apribojimais **(3.2)** gana paprastai įgyvendinti naudojantis žemiau aprašytu parametrinės optimizavimo metodu.

## 3.3 Programinių judesių parametrinis optimizavimas ir optimalaus valdymo sintezė

Aptarsime automatinio sudarymo metodą ir PJ optimizavimą pagal jo karkasą. Ieškomasis PJ turi atitikti apribojimo sąlygas **(3.4)** arba **(3.8)**. Aprašyta aukščiau planavimo metodika ir PJ karkaso sudarymo sąlygos suvedamos prie vienos

$$q_p(t_0) = q_0; \quad q_p(t_T) = q_1 \qquad \textbf{(3.18)}$$

arba prie nuoseklios grupės tokių sąlygų. Kartais prie jų pridedamos apibendrintų greičio vektorių pradinės ir galutinės sąlygos.

$$\dot{q}_p(t_0) = \alpha_0; \quad \dot{q}_p(t_T) = \alpha_1; \qquad \textbf{(3.19)}$$

Kadangi karkasas, pagal kurį sudaromas PJ, atitinka kliūčių apiėjimo reikalavimus **(3.9)**, tai galime teigti (prie kai kurių tikrų sąlygų), kad pats PJ atitinka **(3.9)**. Praktikoje konstruktyvus apribojimai **(3.2)** dažnai išreiškiami taip:

$$c_q \le q_p(t) \le \overline{c}_q; \qquad \textbf{(3.20)}$$

$$c_{\dot{q}} \le \dot{q}_p(t) \le \overline{c}_{\dot{q}}; \qquad \textbf{(3.21)}$$

Taip, PJ sintezės uždavinys išreiškiamas dvigubai diferencijuotą nenutraukiamą funkciją $q_p(t)$, kurį atitinka ribinius apribojimus **(3.18)**, **(3.19)** ir apribojimus **(3.20)**, **(3.21)**.

Suformuluotas uždavinys dažnai turi daug sprendimų. Palyginimui įvairių PJ ir parenkant iš jų optimalių, reikia įvesti kokybės funkcionalą $K[q_p(\cdot)]$. Tada PJ optimizavimo uždavinys susideda iš PJ tipo $q_p^0(t)$ sudarymo, kuris mažina kokybės funkcionalą:

$$K[q_p^0(\cdot)] = \min K[q_p(\cdot)] \qquad \textbf{(3.22)}$$

ir atlieka apribojimus **(3.20)** – **(3.21)**.

Toks optimizavimo uždavinio formulavimas žymiai skiriasi nuo “ekstremalaus metodo” ir jo modifikacijos, kur viskas suvedama prie mažinančio nuoseklumo, atsižvelgiant į kinematinį funkcionalą

$$K(q_p) = \left\| \Phi(q_p) - r_* \right\|. \qquad \textbf{(3.23)}$$

Sprendžiant šį uždavinį naudojami greičiausio nusileidimo algoritmai. Niutono metodo arba labiau efektyvesni rekurentiniai algoritmai, kuriose panaudos pokoordinatinio nusileidimo idėjos. Rezultatas yra kažkoks atbulinis manipuliacinis, lygties **(3.6)** atžvilgių, uždavinio sprendimas. “Ekstremalaus metodo” trūkumas yra manipuliatoriaus dinamikos, aprašytos **(3.6)** – **(3.3)** lygtyse, neįvertinimas.

Aprašytas žemiau PJ parametrinės optimizavimo metodas neturi šito trūkumo. Parodysime pagrindines idėjas šito metodo, manipuliatoriaus PJ optimizavimo “energetinio” kokybės funkcionalo **(3.24),** atžvilgių.

$$K[q_p(\cdot)] = \int_{e_u}^{e_E} (\left\| q_p(t) \right\|^2 + \left\| \dot{q}_p(t) \right\|^2) dt. \qquad \textbf{(3.24)}$$

Pagrindinis šito metodo tikslas yra PJ $q_p(t)$ specialia parametrizacija, kurioje parenkant bazines funkcijas automatiškai atitinka ribinius apribojimus **(3.18)**, **(319)**. Tokio sprendimo idėja: ieškant PJ tinkamų parametrų reikia specialiai stebėti, kad atitiktu ribiniai apribojimai, pavyzdžiui, naudojant L. Pontriagino maksimumo principą. Panagrinėsime vieną iš PJ parametrizacijos metodų.

Ieškosime PJ N-parametrinių funkcijų klasėje

$$q_p(t) = a_0(t) + \sum_{j=1}^{N} \chi_j a_j(t), \qquad \textbf{(3.25)}$$

čia $a_0(t)$ - n-mačia vektorius-funkcija; $\chi_j$ - n-mačiai nežinomi parametrai; $a_j(t)$ - bazinės funkcijos. Tam kad PJ parametrinė funkcija **(3.25)** užtikrintu prie bet kurių $\chi_j$, j=1, …, N, ribinių sąlygų reikalavimus **(3.18)**, $a_0(t), a_j(t),\ j = 1,..., N$ uždėsime sekančius reikalavimus:

$$a_0(t_0) = q_0; \quad a_0(t_T) = q_1; \; a_j(t_0) = a_j(t_T) = 0. \tag{3.26}$$

Tokiems reikalavimams atitinka funkcijos

$$\begin{aligned} a_0(t) &= q_0 + \frac{t - t_0}{T}(q_1 - q_0); \\ a_j(t) &= \alpha(t)\varphi_j(t), \end{aligned} \tag{3.27}$$

čia $\alpha(t_0) = \alpha(t_T) = 0; \quad \varphi_j(t)$ - pilnos sistemos pasirinktos funkcijos. Kaip $\alpha(t)$ ir $\varphi_j(t)$, leidžiantis paprastą schemą arba programinę realizaciją, galime paimti, pavyzdžiui, tokias funkcijas

$$\begin{aligned} &\alpha(t) = (t - t_0)(t_T - t) \; arba \; \alpha(t) = \sin \pi \frac{t - t_0}{T}; \\ &\varphi_j(t) = t^j \; arba \; \varphi_j(t) = \cos \frac{\pi_j (t - t_0)}{T}. \end{aligned} \tag{3.28}$$

PJ parametrai $\chi_j$ **(3.25)** turi atitikti prie visu $t \in [t_0, t_T]$ nelygybių-apribojimų **(3.20)**, **(3.21)**. Sprendimas tokiu nelygybių susijęs su PJ egzistavimu **(6.25)**. Jeigu egzistuoja PJ, atitinkantis apribojimus **(3.20)**, **(3.21)**, tai parenkant bazines funkcijas $a_j(t)$ atsiras atitinkantis jiems parametrizotas PJ **(3.25)** prie kažkokio pakankamai didėlio N. Bet jeigu fiksuotam N ir pasirinktam rinkiniui $\{a_j(t)\}_{j=1}^N$ egzistuoja PJ **(3.25)** , atitinkantis **(6.20)-(6.21**), tai nelygybės-apribojimai išsprendžiami ieškomų parametrų $\chi_1, ..., \chi_N$ atžvilgiu.

Tokiu būdu, PJ sudarymo uždavinys išsprendžiamas pagal nelygybių sistemą, kurią nusako apribojimai **(3.2)** arba **(3.20)**, **(3.21)** parametrų $\chi_1, ..., \chi_N$ atžvilgiu. Praktikoje dažnai daugybės $Q_q, Q_{\dot{q}}$ būna rituliais. Tokiais atvejais apribojimai **(3.1)** , PJ **(3.25)** atrodo taip

$$\left\| a_0(t) + A(t)\chi \right\| \le c_q \text{ prie visu } t \in [t_0, t_T]; \tag{3.29}$$

$$\left\| \dot{a}_0(t) + \dot{A}(t)\chi \right\| \le c_{\dot{q}} \text{ prie visu } t \in [t_0, t_T]; \tag{3.30}$$

čia $c_q, c_{\dot{q}}$ - atitinkamai daugybių $Q_q, Q_{\dot{q}}$ , $\chi = \left| \chi_j \right|_{j=1}^N$, $A(t) = \left| a_j(t) \right|_{j=1}^N$ radiusai . Tarkime, kad nelygybės **(3.29), (3.30)** išsprendžiami prie kažkokių $\chi = \chi_*$ atitinkamai su atsargomis $\delta_q$ ir $\delta_{\dot{q}}$.

Tai reiškia, kad PJ **(3.25)** egzistuoja. Faktiškam PJ sudarymui galima pasinaudoti gradientiniais algoritmais, duodantiems nelygybių **(3.29)**, **(3.30)** sprendimus per galutinį žingsnių skaičių. Tokio algoritmo pavyzdžiu gali būti sekantis rekurentinis sprendimo **(3.29)** algoritmas :

$$\chi_{k+1} = \chi_{л} + \delta_q \left\| A(t_k) \right\|^{-2} A^T(t_k) \frac{q_p(t_k)}{\left\| q_p(t_k) \right\|}, \quad k = 0, 1, ..., N \qquad \textbf{(3.31)}$$

čia $t_k$ – dar vienas nelygybių **(3.29)** pažeidimo momentas, prie $\chi = \chi_k, t \succ t_k, q_p(t_k)$ - vertė **(3.25)** prie $\chi = \chi_k, t = t_k$.

Visiškai analogiškai užrašomas rekurentinis sprendimo **(3.30)** algoritmas. Kompozicija šitų algoritmų po galutinio skaičiaus žingsnių pilnai duoda nelygybių-apribojimų sistemos **(3.29), (3.30)** sprendimą.

Aprašytu PJ galutinių algoritmų parametrų derinimą yra paprasta realizuoti Greita adaptacija ir adaptyvus charakteris pasireiškia nuoseklia žingsnis po žingsnio gerinant PJ parametrus, atsižvelgiant į informaciją apie nelygybių-apribojimų **(3.29)-(3.30)** pažeidimą. Be to tokie algoritmai leidžia naudoti kaip pradinį priartėjimą $\chi_0$ praktiškai bet kokį įvertinimą. Tai daro jos nejautriais išskaičiuojamoms paklaidoms.

Parenkant optimalų PJ pagal klasę **(3.25)** panaudosima optimalumo kriterijų **(3.24)**. Po to kai įstatysime **(3.25)** į **(3.24)** kokybės funkcionalas pavirs kvadratinę funkciją

$$\psi(\chi) = K[q_p(\cdot)] \equiv \frac{1}{2}\langle \chi, H\chi \rangle + \langle b, \chi \rangle + c, \qquad \textbf{(3.32)}$$

čia $H = 2\int_{t_0}^{t_T} [A^T(t)A(t) + \dot{A}^T(t)\dot{A}^T(t)]dt; \quad b = 2\int_{t_0}^{t_T} [A^T(t)a_0(t) + \dot{A}^T(t)\dot{a}_0(t)]dt;$

$$c = \int_{t_0}^{t_T} [\left\| a_0(t) \right\|^2 + \left\| \dot{a}_0(t) \right\|^2]dt.$$

Tokiu būdu PJ optimizavimo variacinis uždavinys sprendžiamas paprastu programavimu: rasti PJ parametrų optimalias vertes **(3.25)** tipo atsižvelgiant į sąlygas

$$\Psi(\chi_1^0, ..., \chi_N^0) = \min_{\chi} \Psi(\chi_1, ..., \chi_N) \qquad \textbf{(3.33)}$$

ir apribojimus **(3.21), (3.30)**.

Skaičiavimo technologija parametriškai optimizuojant PJ, nors ir paprastesnė už optimalaus valdymo klasikinių schemų ir variacinio skaičiavimo teorijos, bet vis dėl to labai sudėtinga ir reikalaujanti daug darbo. Neutralaus programavimo standartiniai metodai reikalauja modifikacijos, susėtos su įvertinimų tam tikrų apribojimų, ir jų realizacija galima tik prieš darbinį etapą, tai yra iki roboto judėjimo pradžios, PJ apskaičiuoja sudėtinga programinė įranga.

Ypatumai pramoninių robotų eksploatacijos metų duoda papildomus apribojimus PJ optimizavimo algoritmams.Pagrindinių iš jų yra sekantis reikalavimai: skaičiavimų paprastumas, galimybė lankstaus perėjimo ir adaptacijos optimizavimo procese. Šitų atžvilgių didelį dėmesį reikia skirti rekurentinėms algoritmams rekurentinio tipo, kurie skirti apitiksliam uždavinio **(3.33)** sprendimui. Sudarysime optimizavimo nelygybė

$$\varphi^0(\chi) = \varepsilon - \left\| grad\Psi(\chi) \right\| = \varepsilon - \left\| H\chi + b \right\| \succ 0, \qquad \textbf{(3.34)}$$

čia $\varepsilon$ - teigiamas skaičius , charakterizuojantis reikiamą tikslumą sprendžiant uždavinį **(3.33)**. Šitą nelygybė išsprendžiama su atsargą $\varepsilon$ prie $\chi = -H^{-1}b$.

Nagrinėjant **(3.34)** ir lyginant su nelygybėmis **(3.29), (3.30),** gausime nelygybių sistemą PJ parametrų aplinkoje. Sėkmingas sprendimas šios sistemos, reiškia, kad optimalus PJ **(3.25)** tipo egzistuoja. Tokiomis sąlygomis, faktiškai gautos nelygybių sistemos sprendimui, taikomi rekurentiniai sueinančių galų algoritmai **(3.31)** tipo (tiksliau, tokiu algoritmų kompozicijos). Įstatant gautus rezultatus $\chi^0$ į formulę **(3.25)**, gausime analitinę išraišką $\varepsilon$ – optimaliam PJ, atitinkančia apribojimus **(3.29), (3.30).**

Žinant optimalų PJ $q_p^0(t)$, labai paprasta sudarytį optimalų programinį valdymą **(3.5),** kuris turi tokį pat sudėtingumą kaip ir dinaminė lygtis **(3.1).** Jam realizuoti reikia tiksliai žinoti $\xi$ parametrus. Bet netgi ir tokių atveju robotas, valdomas pagal optimalią programą **(3.5)**, svyros ant stabilumo ribos. Praktikoje atsirandantis trikdžiai gali išvėsti robotą iš stabilios padėties. Todėl, reikia pripažinti, tad optimalus programinis valdymas **(3.5)** praktiškai neįmanomas.

Atsiranda optimalaus valdymo sintezes būtinumas pagal grįžtamojo ryšio principą. Toks valdymas lengvai sudaromas **(3.5)** pagrindų ir atrodo taip

$$u^0[q,\dot{q}] = A(q,\xi)\ddot{q}_p^0 + b(q,\dot{q},\xi) \qquad \textbf{(3.35)}$$

Tokia lygtis garantuoja optimalaus PJ $q_p^0(t)$ stabilumą, tai yra pasireiškia stabilizavimo efektas pradinių sąlygų atžvilgiu.

Bet tam, kad užtikrinti patikimumą optimalaus PJ $q_p^0(t)$, to nepakanka. Svarbu, kad valdymo dėsnis užtikrintų PJ $q_p^0(t)$ asimptotinį stabilumą. Ieškomasis optimalaus valdymo stabilizavimo dėsnis turį tokią išraišką:

$$u(q,\dot{q}) = A(q,\xi)[\ddot{q}_p^0 + \Gamma_1(\dot{q}-\dot{q}_p^0) + \Gamma_2(q-q_p^0)] + b(q,\dot{q},\xi)\ , \qquad \textbf{(3.36)}$$

čia $\Gamma_1, \Gamma_2$ - matricos grįžtamojo ryšio jėgos padidinimo kanaluose koeficientai, tokie, kad lygties šaknys:

$$\det\left\| I\lambda^2 + \Gamma_1\lambda + \Gamma_2 \right\| = 0 \qquad \textbf{(3.37)}$$

turi neigiamas realias dalis. Įrašant išraišką **(3.36)** į **(3.1)**, gausime sekantį uždaros sistemos lygtį:

$$\ddot{q} = \ddot{q}_p^0 + \Gamma_1(\dot{q}-\dot{q}_p^0) + \Gamma_2(q-q_p^0) \qquad \textbf{(3.38)}$$

Iš šitos išraiškos matyti, kad optimalus PJ $q_p^0(t)$ asimptotiškai stabilus, jeigu robotas valdomas pagal dėsnį **(3.36)**.

Praktikoje parametrai $\xi$ dažniausiai nežinomi. Kartais jie gali keistis pagal nežinomą dėsnį kažkokių daugybių $Q_\xi$ viduje. Tokiais atvejais negalima pasinaudoti optimalaus valdymo dėsniais **(3.5), (3,35)**, **(3.36)** tipo. Atsiranda poreikis optimalaus valdymo adaptyvinio dėsnio sintezės. Adaptyvinio valdymo esmė yra ta, kad idealiame valdymo dėsnyje **(3.36)** nežinomi parametrai $\xi$, pakeičiami jų įvertinimais $\tau$. Tokie įvertinimai automatiškai parenkami pagal adaptacijos algoritmą, kuris užtikrina įvertinimų $\tau$ priartėjimą prie nežinomų parametrų $\xi$. Vykdant adaptacijos algoritmo sintezę dažniausiai leidžiama nauduoti tik grįžtamojo ryšio

signalus, kurie aprašo roboto būseną. Adaptacijos algoritmų konstravimo metodai, o taip pat tokių algoritmų akivaizdžios formulės, aprašyti žemiau.

**Adaptacijos ir optimizavimo algoritmai**

Pramoninei aplinkai charakteringas neapibrėžtumas arba neprognozuojamas roboto parametrų $\xi$ dreifas lemia poreikį papildyti optimalaus valdymo sintezuotus dėsnius **(3.35)** adaptacijos algoritmais (AA). Adaptacijos idėja paprasta. Ji susideda iš automatinės valdymo dėsnio parametru korekcijos. Tuo tikslu nežinomas vektorius $\xi$ pakeičiamas įvertinimu $\tau$, kuris atitinka apibrėžimui AA:

$$\tau(t) = a[\tau_0, q(t), \dot{q}(t)], \quad t \geq t_0 . \qquad \textbf{(3.39)}$$

Aišku iškyla klausimai : kaip pasirinki AA? Ar galima optimizuoti AA?

Adaptyvinio valdymo teorijoje yra keletas diskretinės sintezės ir nepertraukiamų AA sprendimo metodų. Panagrinėsime du metodus: sueinančiųjų rekurentinių galų algoritmų nelygybių sprendimų sistemą ir nepertraukiamos adaptacijos metodą, pagrįsta Liapunovo funkcija. Tam kad pavaizduoti pirmą metodą sudarysime pagalbinių nelygybių sistemą, kurios pažeidimas liudija apie valdymo dėsnio parametrų korekcijos būtinumą.

Pirmiausiai pažymėsime, kad manipuliatorinių robotų dinaminė lygtis **(3.1)** turi pagal parametrus linijinės savybes, tai yra kairiąją lygties **(3.1)** dalį galima įsivaizduoti taip:

$$A(q,\xi)\ddot{q} + b(q,\dot{q},\xi) \equiv G(q,\dot{q},\ddot{q})\xi \quad , \qquad \textbf{(3.40)}$$

čia $G(q,\dot{q},\ddot{q})$ - pasirinkta matrica-funkcija n x N matmenų. Tai svarbi savybė, kuri labai palengvina adaptyvaus valdymo dėsnio algoritminę sintezę. Pereinant prie tokios sintezės, apžvelgsime kita pagalbinių nelygumu sistema:

$$\varphi(\tau,t) \equiv \delta - \left\| u - G(q,\dot{q},\ddot{q})\tau \right\|^2 \succ 0, \quad t \geq t_0 \qquad \textbf{(3.41)}$$

čia $\delta$ - teigiamas skaičius (parametras), u – adaptyvus valdymo dėsnis:

$$u = G[q,\ddot{q},\ddot{q}_p^0 + \Gamma_1(\dot{q} - \dot{q}_p^0) + \Gamma_2(q - q_p^0)]\tau(t). \qquad \textbf{(3.42)}$$

Pažymėsime, kad nelygybės **(3.41)** išsprendžiamos kai $\tau = \xi$ su atsarga $\delta$. Be tuo, funkcija $\varphi$ įlenkta pagal $\tau$. Iš to matyti, kad pagalbinių nelygybių faktinius sprendimus galima naudoti tik AA gradientinio tipo.

Svarbią AA gradientinio tipo klasę sudaro diskretiniai algoritmai

$$\tau(t) = \tau_k \ , \ t \in [t_k, t_{k+1}) \ , \ t_{k+1} = t_k^{`} + \theta, \ \ k = 0,1,2,...;$$
$$\tau_{k+1} = \chi[\tau_k, \tau_{k-1}, ..., \tau_0, \ \ grad\varphi(\tau_k, \tau_k^{`})], \qquad \textbf{(3.43)}$$

čia $t_k^{`}$ - pagalbinių nelygybių pažeidimo pirmasis momentas **(3.41)** kai $\tau = \tau_k$ ; $\theta$ - naujo įvertinimo $\tau_{k+1}$ paskaičiavimo laikas pagal seną įvertinimą $\tau_k$, formulė **(3.43).**

Apibendrinant pavaizduosime diskretinio AA pavyzdį, kuris yra greičiausio nusileidimo metodo analogas. Toks AA atrodo taip:

$$\tau_{k+1} = \tau_k + \frac{\left\| G^T(t_k^{`})\Delta(t_k^{`}) \right\|^2 G^T(t_k^{`})\Delta(t_k^{`})}{\left\| G(t_k^{`})G^T(t_k^{`})\Delta(t_k^{`}) \right\|^2} \qquad \textbf{(3.44)}$$

čia $G(t_k^{`}) \equiv G[q(t_k^{`}), \dot{q}(t_k^{`}), \ddot{q}(t_k^{`})]$, $\Delta(t_k^{`}) = u(t_k^{`}) - G(t_k^{`})\tau_k$.

Reikia atkreipti dėmesį į optimalius galutinai sueinančius AA. Čia kaip adaptacijos kriterijus buvo panauduotas sekantis vietinio optimalumo kriterijus :

$$\left\| \tau_{k+1} - \xi \right\|^2 = \min, \qquad \textbf{(3.45)}$$

čia mažiausia vertė imama pagal AA parametrus. Šio kriterijaus esmė – kaip naują įvertinimą $\tau_{k+1}$ parenkamas įvertinimas, garantuojantis geriausią priartėjimą prie nežinomo vektoriaus $\xi$ optimalaus valdymo AA klases dėsnio **(3.36) (3.43).** Kitaip sakant, optimalus AA, pagal **(3.45),** stengiasi kiekviename žingsnyje kaip galima tiksliau identifikuoti roboto tikrus (bet nežinomus) parametrus $\xi$. Panagrinėsime vietiniai optimalų rekurentinį gradientinį AA. Toks AA atrodo taip:

$$\tau(t) = \tau_k; \quad t \in [t_k, t_{k+1}]; \quad t_{k+1} = t_k^{`} + 0, \quad k = 0, 1, ...;$$
$$\tau_{k+1} = \tau_k + \frac{G^T(t_k^{`})[u(t_k^{`}) - G(t_k^{`})\tau_k]}{\left\| G(t_k^{`}) \right\|^2}. \qquad \textbf{(3.46)}$$

Svarbu pažymėti, kad tokių AA žingsnių skaičius neviršija N. Tai reiškia, kad adaptyvino dėsnio **(3.42)** su optimalių AA žingsnių, parametrų korekcijų skaičius neviršija vektoriaus parametrų $\xi$ dydžio.

Trumpai apžvelgsime nepertraukiamos adaptacijos metodą. Tokiu atveju ieškomų parametrų $\xi$ įvertinimas $\tau(t)$ ir yra diferencialinės adaptacijos lygties sprendimas. Pavyzdžui nepertraukiamas gradientinis AA galį būti toks:

$$\dot{\tau} = G^T(q,\dot{q},\ddot{q})[u - G(q,\dot{q},\ddot{q})\tau]; \quad \tau(t_0) = \tau_0, \quad t \geq t_0. \qquad \textbf{(3.47)}$$

Pažymėsime, kad AA **(3.47)** dešiniosios pusės išraiška yra lygi $1/2\, grad\, \varphi(\tau,t)$, čia $\varphi$ apibudina santykį **(3.41)**. Paprasta parodyti (panaudojant Liapunovo funkciją) , kad įvertinimas $\tau(t)$ priklausomai nuo AA **(3.41)** sutapatinama pagal optimalaus valdymo dėsnį **(3.36)** su ieškomų parametrų vektoriumi $\xi$.

Nepartraukiamus AA paprasta realizuoti analoginiais keitikliais ar specialiosiomis skaičiuoklėmis. Realizuojant diskretinius AA **(3.44), (3.46)** verta naudoti mikroprocesorius.

Reikia pažymėti, kad nagrinėti AA aplamai tiksliai nesuranda nežinomų parametrų $\xi$. Bet to ir nereikia: parenkant "padidinimo koeficientus", $\Gamma_1, \Gamma_2$ matricas ir parametrus $\delta, \theta$, visada galima pasiekti realaus roboto judėjimo $q(t)$ ir PJ $q_p^0(t)$ reikiamo tikslumo. Toks faktas patvirtina, kad optimaliam roboto judėjimui, jeigu nėra tikslios informacijos apie dinaminės lygties **(3.1)** parametrus , tikslių parametrų vertės nebūtinos.

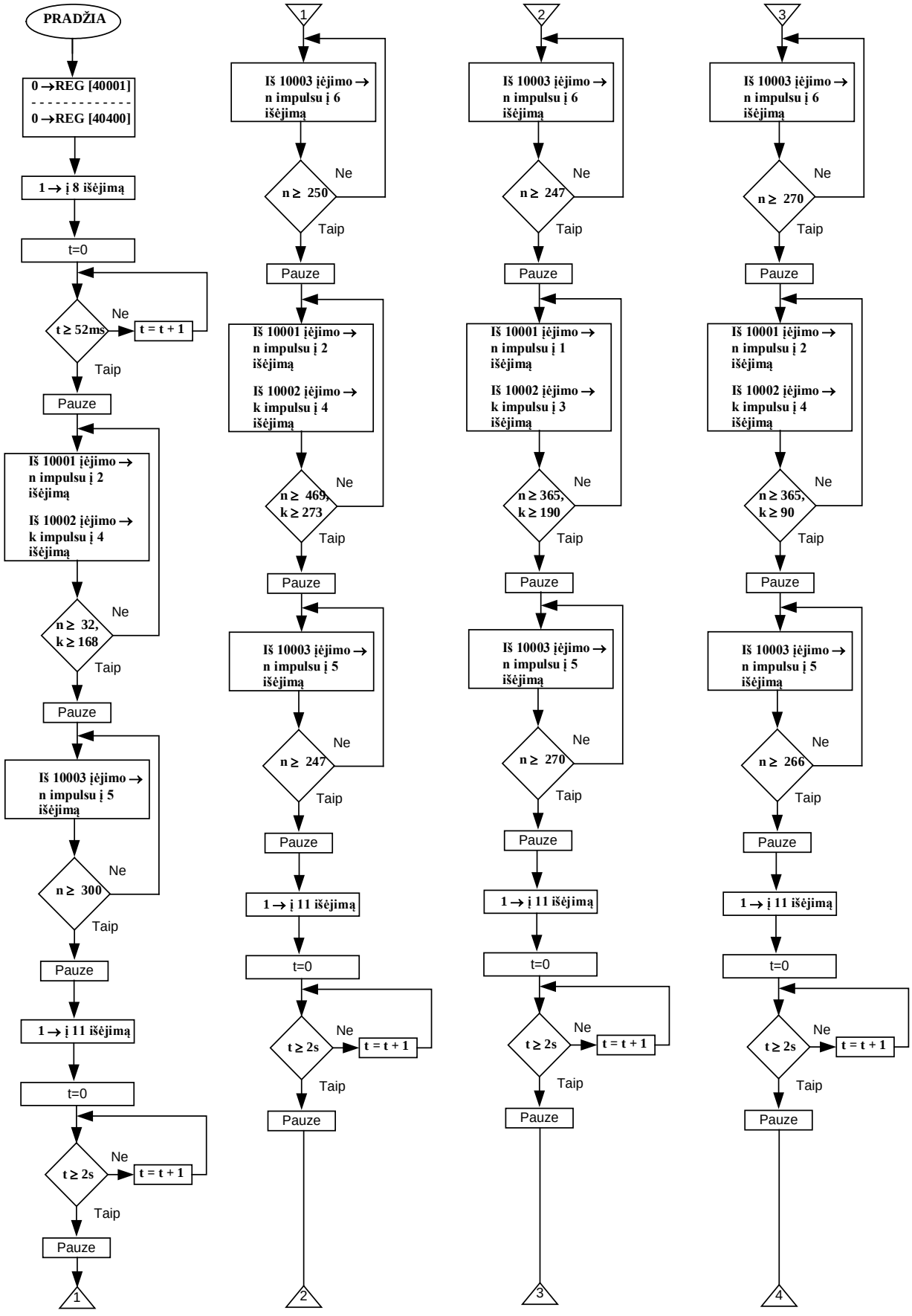
Tokiu būdu koreguojant optimalaus dėsnio parametrus pagal sintezuotus AA, faktiškai garantuojamas roboto optimalus PJ. Pažymėsime, kad gauti tokį rezultatą nežinomoje aplinkoje be adaptacijos praktiškai neįmanoma. To ir pasižymi praktiškas adaptacijos efektyvumas, kai optimizuojamas valdymas, neturint informacijos apie roboto charakteristikas ir eksploatacijos sąlygų **[19]**.

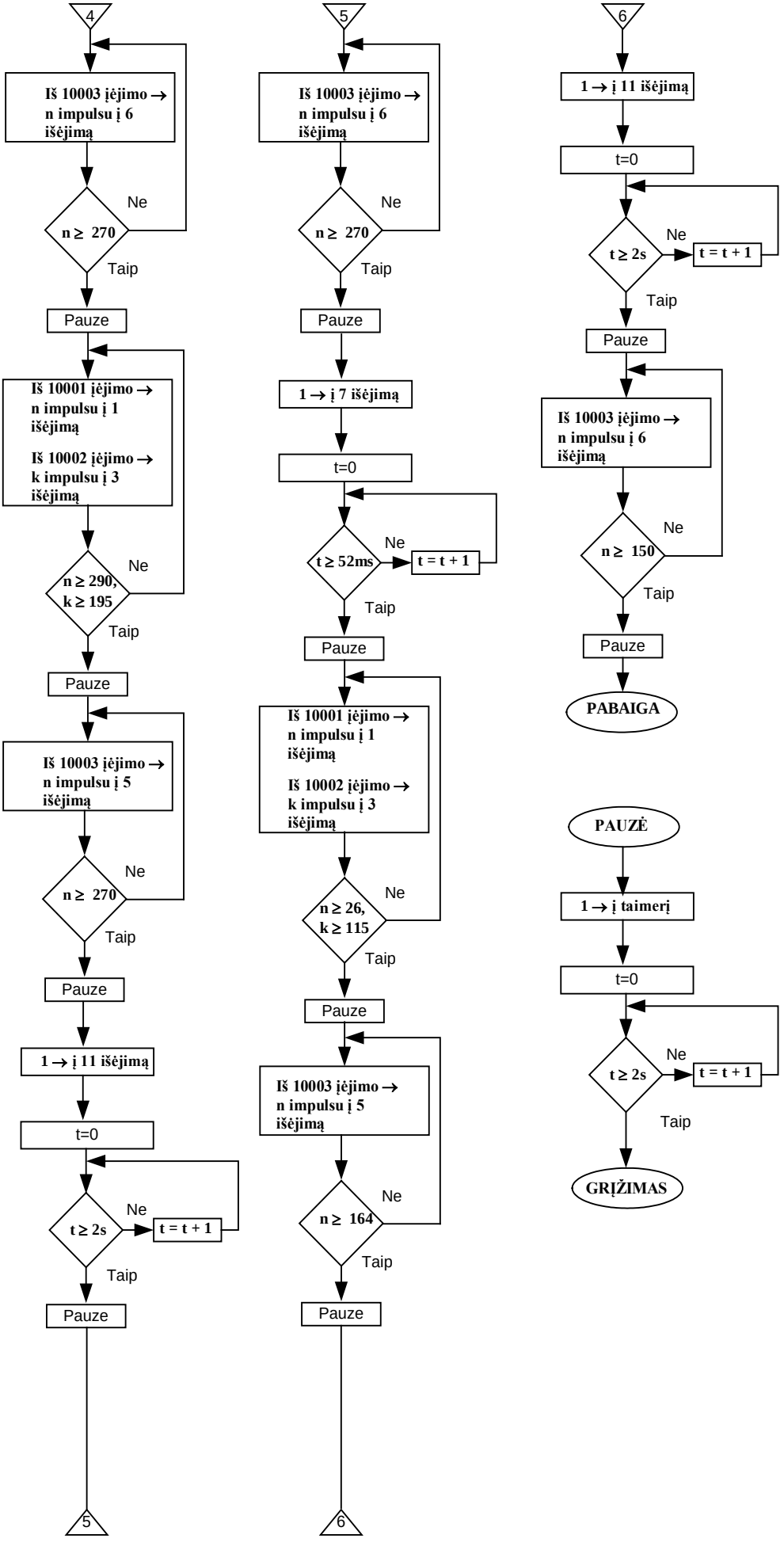
## 3.4 Roboto URTK valdymo algoritmo optimizavimas

Optimalių sprendimų paieška taikant paieškos optimizavimo metodą yra laipsniško artėjimo prie optimalaus taško procesas, kuris sudaromas panaudojant eksperimento rezultatus, gautus optimizuojant objektą. Informacijos, gautos eksperimento būdu, analizė leidžia parinkti paieškos žingsnio kryptį ir dydį, įvertinti ekstremalias objekto savybes neapibrėžtumo sąlygomis. Pagrindinė paieškos optimizavimo ypatybė yra ta, kad judėjimo kryptis link tikslo nustatoma pagal išmatuotas tikslo funkcijos vertes tam tikro eksperimentinio plano taškuose. Paieškiniai eksperimentai leidžia papildyti trūkstamą informaciją apie objektą ir suteikia paieškinės optimizavimo metodams universalumo.

Šito darbo tikslas optimizuoti roboto valdymo sistema. Vienas iš  būdu tai atlikti tai optimizuoti roboto valdymo algoritmą. Kaip pavyzdį paimsime mano bakalauro baigiamajame darbe sudaryta roboto URTK valdymo algoritmą ir optimizuosime jį pagal laiką.  Ankstesnis roboto URTK valdymo algoritmas buvo sudarytas taip, kad pagal kiekvieną koordinatę robotas judėjo atskirai ir bendras roboto judėjimo laikas buvo didėlis. Pagrindinė idėja optimizuojant valdymo algoritmą pagal laiką, tai valdyti kelias koordinates tuo pačių metų. Kadangi roboto judėjimas labai apribotas ir egzistuoja kliūtys, kurias būtinai reikia įvertinti, buvo priimta valdyti kartu X ir Y koordinates, o koordinatę Z  atskirai. Neoptimizuotą algoritmą sudaro  32 ciklai, optimizuotas algoritmas sumažėjo iki 26 ciklų. X ir Y koordinatės yra didžiausios. Judėjimo laikas pagal šias koordinates sudaro apie 70 procentų bendro roboto judėjimo laiko. Valdant koordinates X ir Y tuo pačių metų bendras roboto judėjimo laikas sumažėja iki 30 procentų.

## Optimizuotas roboto URTK valdymo algoritmas




4
Iš 10003 įėjimo → n impulsu į 6 išėjimą
n ≥ 270
Ne
Taip
Pauze
Iš 10001 įėjimo → n impulsu į 1 išėjimą
Iš 10002 įėjimo → k impulsu į 3 išėjimą
n ≥ 290, k ≥ 195
Ne
Taip
Pauze
Iš 10003 įėjimo → n impulsu į 5 išėjimą
n ≥ 270
Ne
Taip
Pauze
1 → į 11 išėjimą
t=0
t ≥ 2s
Ne
t = t + 1
Taip
Pauze
5
5
Iš 10003 įėjimo → n impulsu į 6 išėjimą
n ≥ 270
Ne
Taip
Pauze
1 → į 7 išėjimą
t=0
t ≥ 52ms
Ne
t = t + 1
Taip
Pauze
Iš 10001 įėjimo → n impulsu į 1 išėjimą
Iš 10002 įėjimo → k impulsu į 3 išėjimą
n ≥ 26, k ≥ 115
Ne
Taip
Pauze
Iš 10003 įėjimo → n impulsu į 5 išėjimą
n ≥ 164
Ne
Taip
Pauze
6
6
1 → į 11 išėjimą
t=0
t ≥ 2s
Ne
t = t + 1
Taip
Pauze
Iš 10003 įėjimo → n impulsu į 6 išėjimą
n ≥ 150
Ne
Taip
Pauze
PABAIGA
PAUZĖ
1 → į taimerį
t=0
t ≥ 2s
Ne
t = t + 1
Taip
GRĮŽIMAS

## 3.5 Roboto valdymo paprogramė (programavimo kalbą C++ versija 3.11)

Sukurta roboto valdymo paprogramė tinka ne tik anksčiau minėtam robotui URTK bet ir kitiems panašaus veikimo pramoniniams robotams. Tokios paprogramės privalumas tas, kad aprašant roboto judėjimą, nereikia aprašinėti kiekvieno judesio atskirai, užtenka pasirinkti tik reikiamas koordinates. Naudojant šitą paprogramę konkrečiai roboto judėjimo programai sudaryti, reikia tinkamai suderinti ryšį tarp kompiuterio, mikrovaldiklio ir roboto valdymo sistemos. Mikrovaldiklį galima pasirinkti iš anksčiau minėtų, firmos Modicon gaminamų, mikravaldiklių.

```
//-----------------------------------------------
// CheckOnline – paprogramė skirta valdymo sistemos pajungimui patikrinti
//     int CheckOnline(void)
//      Input:          none
//      Output: 1 – Valdymo sistema pajungta,
//       kitaip 0
//      Desc:           Valdymo sistemos pajungimo patikrinimas
//-----------------------------------------------
int CheckOnline(void)
{
if (ReadReg(RC) == cOnline) return 1;
else return 0;
}
//-----------------------------------------------
//     SetDvgRegs – paprograme skirta įrašyti į variklių valdymo registrus
//     tinkamus parametrus
//      int SetDvgRegs(byte dvg0, byte dvg1)
//      Input:          dvg0, dvg1 – šitų kintamųjų vertės užrašomos į
//                                   variklių registrus
//      Output: visada 1.
//      Desc:   Valdymo sistemos registru valdančiu variklius nustatymas
//-----------------------------------------------
int SetDvgRegs(byte dvg0, byte dvg1)
{
WriteReg(RC, cSetDvgRegs);
// priverstinis variklių apvijų žadinimas
if ((dvg0 != 0) || ((dvg1 & 0x0F) != 0)) dvg1 |= 0x80;
WriteReg(PA, dvg1);
WriteReg(PA, dvg0);
return 1;
}

//-----------------------------------------------
//     CountDvg0 – paprogramė skirta pasirinkti norimas koordinačių kryptis
//      byte CountDvg0(int x, int y, int z, int w, byte dvg0old)
//      Input:  x, y, z, w – pasirinktos kryptis atitinkantis roboto judėjimo
//              kryptims : +1 atitinka judėjimui nuo pradinio taško iki
//              galutinio, -1 atitinka judėjimui nuo galutinio taško iki
//              pradinio,
//              0 - sustojimas, bet kokia kita vertė atitinka pradinės
//              krypties judėjimui
//              dvg0old – ankstesnė vertė įrašoma į atitinkamą valdymo
```

```
//             sistemos registrą
//      Output: Suformulota vertė, kuria gali buti panauduota funkcijoje
//             SetDvgRegs().
//      Desc:   Roboto valdymo registro vertės DVG0 formavimas, kuris valdo //
variklių judėjimo kryptis
//-------------------------------------------------
byte CountDvg0(int x, int y, int z, int w, byte dvg0old)
{
byte res = 0;
switch (x) {
case -1:       SetBit(&res, aXmBit); break;
case 0: break;
case 1: SetBit(&res, aXpBit); break;
default:       if (TestBit(&dvg0old, aXmBit)) SetBit(&res, aXmBit);
if (TestBit(&dvg0old, aXpBit)) SetBit(&res, aXpBit);
}
switch (y) {
case -1:       SetBit(&res, aYmBit); break;
case 0: break;
case 1: SetBit(&res, aYpBit); break;
default:       if (TestBit(&dvg0old, aYmBit)) SetBit(&res, aYmBit);
if (TestBit(&dvg0old, aYpBit)) SetBit(&res, aYpBit);
}
switch (z) {
case -1:       SetBit(&res, aZmBit); break;
case 0: break;
case 1: SetBit(&res, aZpBit); break;
default:       if (TestBit(&dvg0old, aZmBit)) SetBit(&res, aZmBit);
if (TestBit(&dvg0old, aZpBit)) SetBit(&res, aZpBit);
}
switch (w) {
case -1:       SetBit(&res, aWmBit); break;
case 0: break;
case 1: SetBit(&res, aWpBit); break;
default:       if (TestBit(&dvg0old, aWmBit)) SetBit(&res, aWmBit);
if (TestBit(&dvg0old, aWpBit)) SetBit(&res, aWpBit);
}
return res;
}
//-------------------------------------------------
//      WriteReg – paprogramė skirta informacijai įrašymui į valdymo sistemos
// registrus
//      void WriteReg(byte reg, byte value)
//      Desc:   siunčią kintamojo vertę į valdiklio registrą
//-------------------------------------------------
void WriteReg(byte reg, byte value)
{
byte ConfigByte = cPassiveState;

outp(LptConfig, ConfigByte);
switch (reg) {
case PA: SetBit(&ConfigByte, A0); SetBit(&ConfigByte, A1); break;
case PB: SetBit(&ConfigByte, A0); break;
case PC: SetBit(&ConfigByte, A1); break;
case RC: break;
}
ClrBit(&ConfigByte, WR);
ClrBit(&ConfigByte, DIR);
outp(LptData, value);
outp(LptConfig, ConfigByte);
delay(1);                       // Užlaikymas 1 ms
ConfigByte = cPassiveState;
outp(LptConfig, ConfigByte);
}
```

```
//-------------------------------------------------
//      ReadReg - paprogramė skirta informacijos skaitymui iš valdymo
// sistemos registrų
//      byte ReadReg(byte reg)
//      Desc:   skaito baitus iš registro
//-------------------------------------------------
byte ReadReg(byte reg)
{
byte ConfigByte = cPassiveState;
byte d;

outp(LptConfig, ConfigByte);
outp(LptData, 0xFF);

switch (reg) {
case PA: SetBit(&ConfigByte, A0); SetBit(&ConfigByte, A1); break;
case PB: SetBit(&ConfigByte, A0); break;
case PC: SetBit(&ConfigByte, A1); break;
case RC: break;
}
SetBit(&ConfigByte, RD);
SetBit(&ConfigByte, DIR);

outp(LptConfig, ConfigByte);
delay(1);                       // Užlaikymas 1 ms
d = inp(LptData);
ConfigByte = cPassiveState;
outp(LptConfig, ConfigByte);

return d;
}
-------------------------------------------------------------------------
```

# 4. OPTIMALIOS PAGAL GREITAIGIŠKUMĄ SISTEMOS SINTEZĖ

Sakykime, objektas apibūdinamas diferencialine lygtimi:

$$T\frac{d^2X}{dt^2}+\frac{dX}{dt}=U. \qquad \textbf{(4.1)}$$

Reikia rasti algoritmą, kuris pervestų objektą iš pradinės būsenos X = 0, $\dot{X}$ = 0, (pradinis roboto rankos trajektorijos taškas), kai t = 0, į būseną $X = X_0$, $\dot{X} = 0$ (galutinis trajektorijos taškas) per mažiausią laiką; valdymo poveikis apribojamas $|U| \le U_{\max}$.

Sprendimas: Pasižymėję $\frac{dX}{dt} = Y$, **(4.1)** lygtį užrašysime sistemos pavidalu:

$$\begin{aligned}\frac{d}{dt} &= Y,\\ \frac{dY}{dt} &= \frac{1}{T}(U-Y).\end{aligned} \qquad \textbf{(4.2)}$$

Sudarysime funkciją:

$$H = \Psi_1 \frac{dX}{dt} + \Psi_2 \frac{dY}{dt} = \Psi_1 Y + \Psi_2 \frac{1}{T}, \qquad \textbf{(4.3)}$$

Sudarysime funkcijas $\Psi_1$ ir $\Psi_2$ iš Hamiltono lygties:

$$\frac{d\Psi_1}{dt} = -\frac{\partial H}{\partial X} = 0; \qquad \frac{d\Psi_2}{dt} = -\frac{\partial H}{dY} = -\Psi_1 + \Psi_2 \frac{1}{T}.$$

Iš čia $\Psi_1 = -C_1$, $\Psi_2 = C_2 e^{\frac{t}{T}} + C_1 T$. Įrašę vertes $\Psi_1$ ir $\Psi_2$ į **(4.3)** gausime:

$$H = (C_2 e^{\frac{t}{T}} + C_1 T)\frac{1}{T} U - C_2 e^{\frac{t}{T}} + C_1 T)\frac{1}{T} Y + C_1 T. \qquad \textbf{(4.4)}$$

Pagal maksimumo principą, kad procesas būtų optimalus, reikia bet kuriuo momentu parinkti tokią U vertę, kad H dydis būtų didžiausias. Nagrinėjant pirmąjį dėsnį **(4.4)**, nesunku pastebėti, kad norint jog H = H max, būtina parinkti $|U| \le U_{\max}$. Be to, valdymo poveikis turi keisti ženklą tiek kartų, kiek kartų keičia funkciją $\Psi_2$ :

$$U = U_{\max} sign \Psi_2. \qquad \textbf{(4.5)}$$

Kadangi funkcija $\Psi_2$ , esant bet kurioms $C_1$ ir $C_2$ vertėms laiko atkarpoje t > 0, keičia ženklą ne daugiau kaip vieną kartą , tai optimaliame valdymo procese bus ne daugiau kaip vienas perjungimas nuo $U = U_{\max}$ prie $U = -U_{\max}$ arba atvirkščiai.

Vadinasi, optimali pagal greitaeigiškumą sistema bus relinė, turinti ypatingą valdymo poveikio U kitimo dėsnį.

Sistemos valdančiosios dalies sintezei galima panaudoti **(4.5)** priklausomybę, t.y. U kitimą, kaip laiko funkciją, arba kitimo dėsnį, kaip priklausomybę nuo fazinių koordinačių.

Anksčiau gautų rezultatų pagrindu panagrinėsime sistemos valdančiosios dalies sintezę, taikydami fazinę plokštumą.

Lygtį (3.1) užrašysime sistemos paklaidos atžvilgiu:

$$T\frac{d^2 x}{dt^2} + \frac{dx}{dt} = -U. \qquad \textbf{(4.6)}$$

čia $x = X_0 - X$.

Tada optimalaus valdymo uždavinys formuluojamas : būtina sumažinti sistemos paklaidą nuo $x = X_0$ , $\dot{x} = 0$ dydžio per mažiausią laiką, esant apribojimui $|U| \le U_{\max}$.

Raskime fizinių trajektorijų lygtis. Šiam tikslui (4.6) lygtį pateikiame kaip sistemą:

$$\begin{aligned} \frac{dx}{dt} &= y, \\ \frac{dy}{dt} &= -\frac{1}{T}(u + y). \end{aligned} \qquad \textbf{(4.7)}$$

Padalinę pirmą (4.7) sistemos lygtį iš antros ir suintegravę, gausime:

$$\int dx = -T \int \frac{y}{u + y} dy,$$

iš čia gauname fazinės trajektorijos lygtį:

$$x = -Ty + Tu \ln|u + y| - Tu \ln|u|. \qquad \textbf{(4.8)}$$

Nustatysime integravimo konstantą, panaudodami pradines sąlygas $x = -X_0, y = 0$, ir jos vertę įrašysime į (4.8) lygtį:

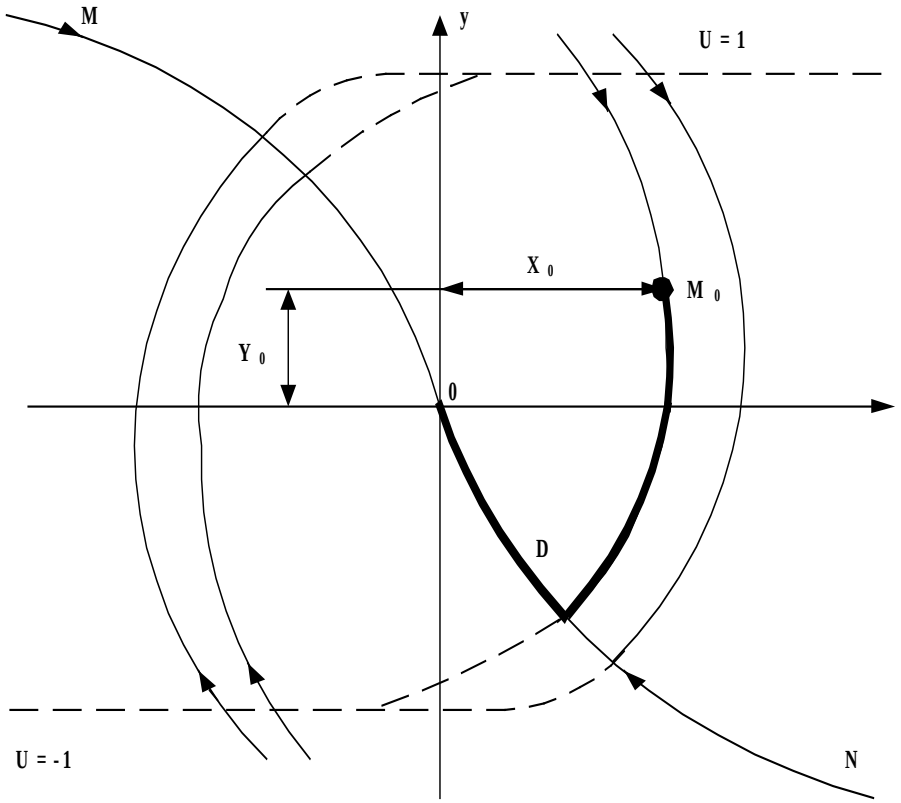
$$x = \pm X_0 - Ty + Tu \ln|u + y| - Tu \ln|u|. \qquad \textbf{(4.9)}$$

Anksčiau buvo nustatyta, kad optimaliame valdymo procese poveikis U įgyja ribines vertes $\pm U_{\max}$ , be to, šiam objektui ženklas gali pasikeisti ne daugiau kaip vieną kartą. Sakykime, kad $\pm U_{\max} = \pm 1$. Tada iš (4.9) gauname:

$$x = X_0 - Ty + T \ln|1 + y|, kai\ u = 1; \qquad \textbf{(4.10)}$$

$$x = -X_0 - Ty - T \ln|y - 1|, kai\ u = -1. \qquad \textbf{(4.11)}$$

Vadinasi fazinis sistemos vaizdas **(4.1 pav.),** sudarytas iš dviejų pusplokštumių. Kiekvieną iš jų atitinka tam tikras fazinės trajektorijos tipas, apibūdintas **(4.10)** ir **(4.11)** lygtimis.




**4.1 pav. Fazinis sistemos vaizdas**

Riba tarp šių plokštumų vadinama perjungimo linija ir sudarytas iš **(4.10)** trajektorijos dalies, kai $X_0 \prec 0, y \prec 0,$ vedančių prie koordinačių pradžios. Perjungimo linijos lygties išraiška:

$$\hat{y} = -Ty + Tsign\ y \ln|sign\ y + y|. \qquad \textbf{(4.12)}$$

Iš laisvai pasirinkto taško $M_0$ (**4.1 pav.**) sistemos judėjimo procesas vyksta trajektorija $M_0D$, kai valdymo signalas U = 1. Taške D, kuris yra ant perjungimo linijos M0N, U signalo ženklą būtina pakeisti priešingu, tada minėtasis taškas judės koordinačių pradžios link trajektorija D0. Toks procesas užsibaigia per apribotą laiką, kuris pagal maksimumo principą yra mažiausiais iš visų galimų pasirinktos sistemos perėjimų iš būsenos $M_0(x = X_0, y = Y_0)$ į nusistovėjusią būseną 0 (x = 0, y = 0) **[6].**

# 5. ATVIROS KVAZIOPTIMALIOS VALDYMO SISTEMOS SINTEZĖ

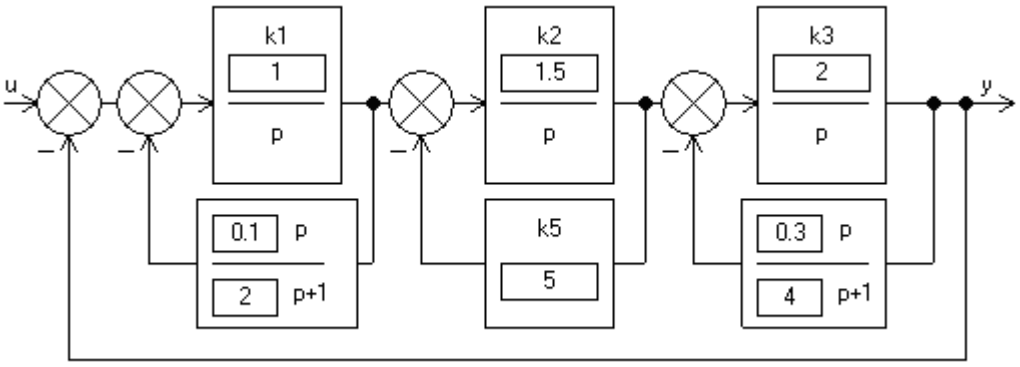
Programinis paketas *Kvazio1* skirtas kvazioptimalių ir kintamos struktūros sistemoms projektuoti tais atvejais, kai objekto matematinis modelis sudėtingas, ar jo išvis nėra, kai klasikinių sintezės metodų taikyti neįmanoma. Kaip pavyzdį galime paimti robotą, kuris veikia aplinkoje, kurios poveikis jam nežinomas arba jo negalima nustatyti.

Programinio paketo *Kvazio1* taikymo sritys – kvazioptimalių ir kintamos struktūros sistemų projektavimas naudojant algoritminius sistemų sintezės metodus.

Šis paketas veikia *Windows* aplinkoje, turi daug didesnių galimybių sprendžiant įvairius kvazioptimalių ir kintamos struktūros sistemų sintezės uždavinius algoritminiais metodais. Taikant algoritminius sistemų sintezės metodus, įvairių AVS sintezės uždavinių formulavimas turi bendrą metodinį pagrindą – visi uždaviniai formuluojami paieškos optimizavimo uždavinių forma. Šių uždavinių sprendimo rezultatai – kvazioptimalūs valdymo poveikio dėsniai, reguliatorių, koregavimo grandžių struktūros ir parametrų kitimo dėsniai ir t.t.

Jeigu objekto matematinis modelis yra žinomas, tai sprendžiant valdymo sistemos sintezės uždavinį, pirmiausia įvedama objekto perdavimo funkcija, toliau sudaroma reikiama valdymo sistemos struktūrinė schema ir optimizavimo kontūras.

Kaip pavyzdį paimsime valdymo sistemą, pateiktą **5.1pav**.




**5.1** pav.Valdymo sistema

**Atvirųjų kvazioptimalių valdymo sistemų sintezės uždavinių formulavimas**

Formuluodami atvirosios kvazioptimalios valdymo sistemos sintezės uždavinį, pristatysime objektą vektorine diferencialine lygtimi:

$$y = f(y,u); \qquad \textbf{(5.1)}$$

čia: $y$ – n-matis būsenos vektorius;

$u$ – n-matis valdymo vektorius.

Vektorinė funkcija $f$ neduota arba yra sudėtingos formos, todėl optimalaus valdymo uždavinio negalima spręsti analitiniu būdu.

Tarkime, kad esant pasirinktoms kraštinėms sąlygoms ir apribojimams egzistuoja optimalus valdymo dėsnis *u(t),* kuris suteikia funkcionalui

$$I = \int_{t_0}^{t_f} f_0(y,u)dt; \qquad \textbf{(5.2)}$$

ekstremumą.

Tada apytikrę vertę $u^*(t)$ galima surasti paieškinio optimizavimo (identifikavimo) procese šiuo būdu. Valdymo intervale $t_0 \le t \le t_f$, kai $t_0=0$, naudojant vektoriaus $u$ komponentų diskretines vertes

$$\begin{aligned} &u_1[jT],\ j = 0,...,r-1,\\ &.....\\ &u_m[jT],\ j = 0,...,r-1, \end{aligned} \qquad \textbf{(5.3)}$$

įvedamas $k = mr$ – matis vektorius $x$:

$$x = \{x_1 = u_1[0], x_2 = u_1[IT],..., x_{k-1} = u_m[(r-2)T], x_k = u_m[(r-1)T]\}; \qquad \textbf{(5.4)}$$

čia $T$ – kvantavimo periodas, $T = t_f/r$.

Naudojant vektoriaus $x$ komponentus, formuojamos slenkstinės arba kitokios iš tiesinių intervalų sudarytos funkcijos:

$$\begin{aligned} &u_1 = (x_i;t), \quad 0 \le t \le t_f,\\ &.....\\ &u_m = u_m(x_i;t), \quad 0 \le t \le t_f. \end{aligned} \qquad \textbf{(5.5)}$$

Slenkstinis funkcijos *u(t)* formavimas, naudojantis vektoriaus komponentais $x_i$, parodytas **5. 2** pav., kai *m=1*.

Taigi vektoriaus *x* komponentai sudaro valdymo dėsnį *u(x,t)* intervale $t_0 \le t \le t_f$.

Tada optimalaus valdymo uždavinys tampa paieškos optimizavimo uždaviniu. Reikia surasti vektorių $x^*$, garantuojantį funkcionalo

$$I(x) = I[y, u(x,t)], \quad 0 \le t \le t_f . \tag{5.6}$$

mažiausią vertę, įskaitant kraštines sąlygas $y(t_0)$, $y(t_f)$ ir laikantis apribojimų

$$h_i[y, u(x,t)] = 0, \quad i = 1,..., p < k, \tag{5.7}$$

$$g_j[y, u(x,t)] \ge 0, \quad j = 1,..., q, \tag{5.8}$$

$$x \in \Omega_x ; \tag{5.9}$$

čia *I*, *h*, *g* – valdymo kokybės rodikliai (reguliavimo laikas $t_r$, didžiausia dinaminė nuokrypa $\sigma$, valdymo paklaida $\Delta y$ ir kt.).

Šis metodas gali būti panaudotas spręsti plačios klasės valdymo uždaviniams, kurie neišsprendžiami klasikiniais optimalaus valdymo ir variacinio skaičiavimo metodais. Būtina sąlyga uždaviniui spręsti yra eksperimentinis funkcionalo **(5.6)** ir apribojimo funkcijų **(5.7), (5.8)** verčių nustatymas duotiems vektoriaus *x* komponentams.

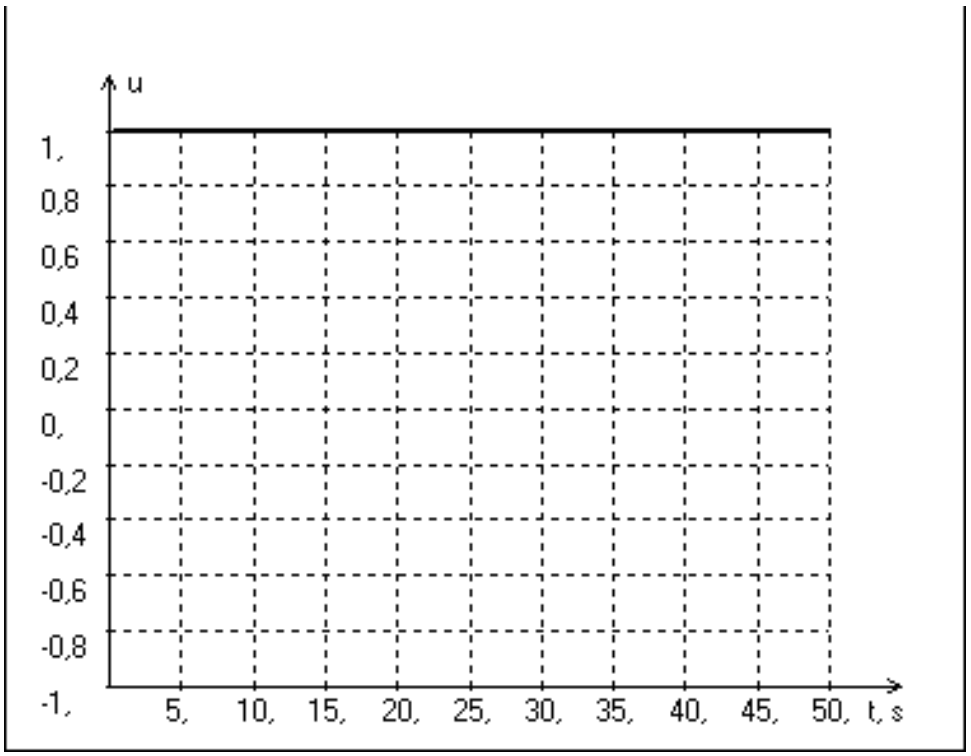
Funkcijos **(5.3)** priimamos įvairiais būdais. Jas pritaikius konkrečiai uždavinių klasei galima sumažinti komponentų $x_i$ skaičių ir padidinti $u^*(t)$ paieškos proceso efektyvumą. Tiesiniams optimalaus valdymo uždaviniams funkcija *u(t)* gali būti priimta komponentais $x_i$, kurie turi perjungimo momentų vertę, jeigu valdymo poveikis kinta nuo vienos ribinės vertės $-u_m$ iki kitos $u_m$, o galutinė jo vertė yra $u_f$.

Jeigu funkcinių apribojimų yra keletas, paieškinio optimizavimo uždavinys formuluojamas baudos funkcijų metodu. Uždavinys **(5.6) – (5.9)** gali būti užrašytas taip:

$$I_I(x) = I(x) + \sum_{i=1}^{p} \Psi_i h_i^2[y, u(x,t)] - \sum_{j=1}^{q} \varphi_j g_j[y, u(x,t)] \times \left\{1 - signg_j[y, u(x,t)]\right\} \to \min \tag{5.10}$$

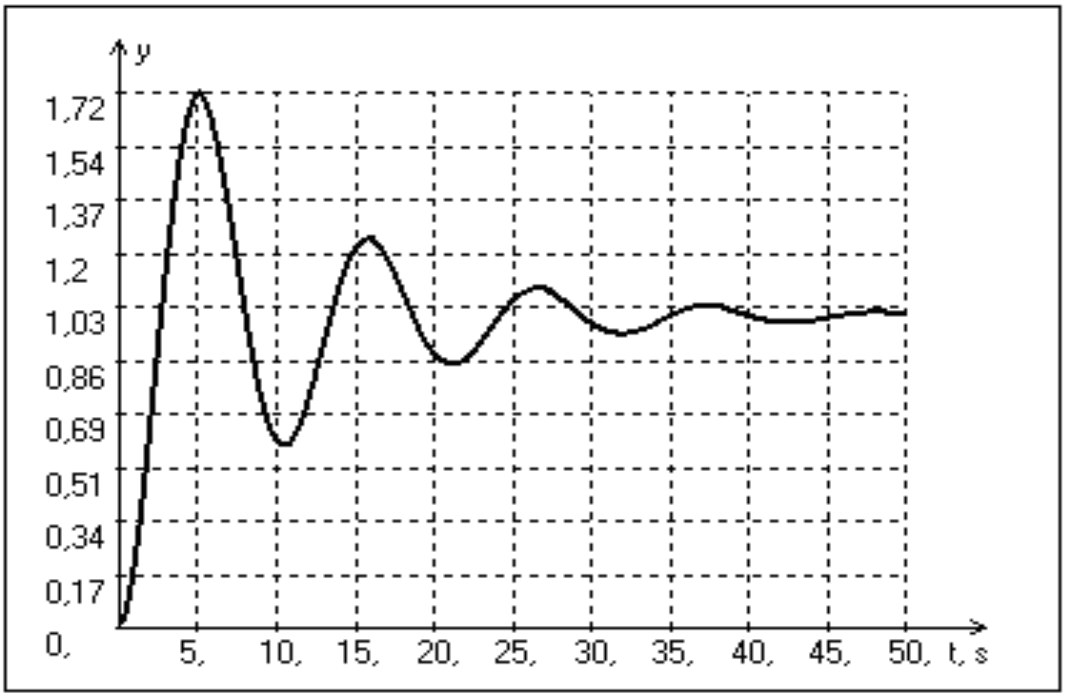
$$x \in \Omega_x ; \tag{5.11}$$

Pasirinkus meniu punktą *Poveikis*, į ekraną atskirame lange išvedamas įėjimo signalo kitimo grafikas **(5.2 pav.).**




**5.2 pav**. Poveikio langas

Schemos pereinamojo proceso kreivė iškviečiama meniu komanda: *Procesas*/*Pereinamojo proceso skaičiavimas* arba mygtuku. Pereinamojo proceso, kai įėjimas lygus 1, pavaizduotas **5.3 pav.**




**5.3 pav**. Analizuojamo objekto pereinamasis procesas

Iš grafiko galima spręsti, kokį reikia nustatyti modeliavimo laiką ir kitus duomenis.

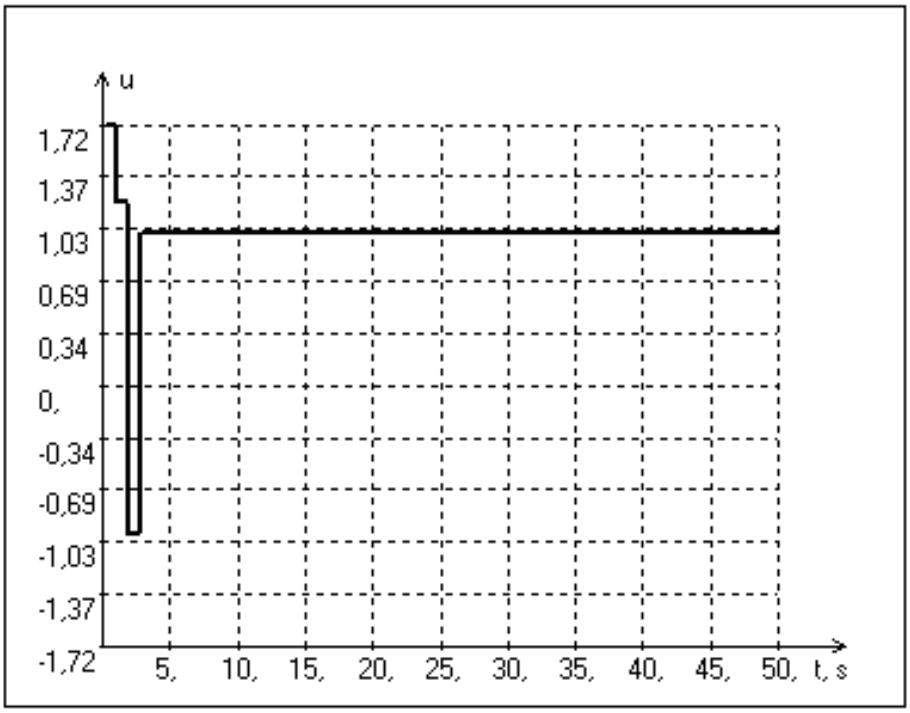
Spendžiant AVS sintezės uždavinius, optimizavimas gali būti atliekamas pagal kelis simpleksinės paieškos algoritmus: su uždraustu grįžimu, su būsenų atpažinimu, aktyvios-pasyvios paieškos ir modifikuotu Nelderio-Mido. Pasirinkus paieškos algoritmą, į ekraną išvedamas pradinių optimizavimo parametrų nustatymo langas. Jame pasirinkami tokie parametrai:

1) Komponentų skaičius. Čia pasirinkamas simplekso viršūnių skaičius. Jis gali būti pasirinktas intervale nuo 2 iki 280. Nuo komponentų skaičiaus *k* priklauso simplekso viršūnių skaičius *k+1*. Paprastai nerekomenduojama rinktis labai daug komponentų, nes labai pailgėja skaičiavimo trukmė ir sumažėja simplekso nueitas kelias. Tipiniams uždaviniams spręsti pakanka iki 10 komponentų.
2) Ribinės komponentų vertės. Pagal šitas vertes pasirinkama apatinė ir viršutinė ribos, kurios negali viršyti komponentas. T.y., šiomis vertėmis apribojama simplekso judėjimo erdvė. Apatinė riba negali būti lygi ar viršyti viršutinės ribos.
3) Nusistovėjusi išėjimo signalo vertė. Čia nurodoma kokia yra (ar turėtų būti) pageidaujamo pereinamojo proceso nusistovėjusi vertė.
4) Valdymo trukmė. Apytikrė valdymo trukmė pasirinkama kaip pereinamojo proceso trukmės dalis. Jo metu atliekama kvazioptimalaus ar kintamos struktūros valdymo dėsnio paieška. Pasibaigus valdymo trukmei, sistemos įėjimo signalas arba koeficiento vertė prilyginama iš anksto pasirinkta jai vertei.
5) Dėsnio formavimo būdas. Galima pasirinkti norimą poveikio ar parametro kitimo dėsnio formavimo būdą: naudojant perjungimo momentus arba diskretines dėsnio funkcijas. Pasirinkus dėsnio formavimą ir pasirinkant perjungimo momentus reikia įvesti didžiausią galimą valdymo dėsnio vertę.

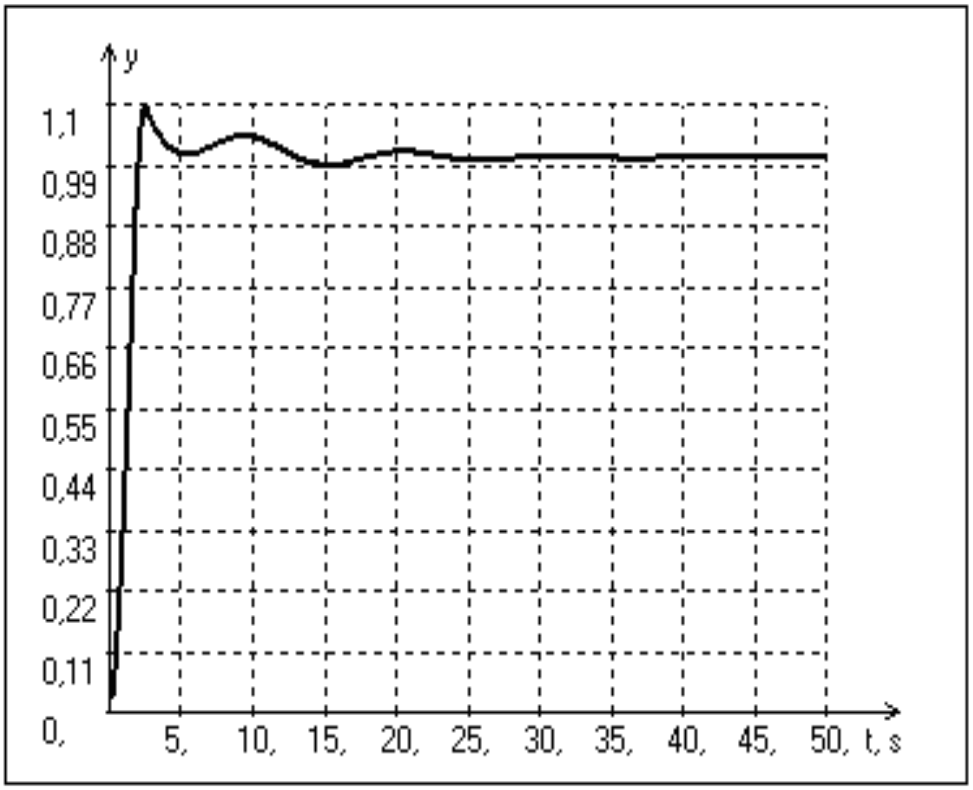
6) Didžiausia dinaminė nuokrypa. Leistina didžiausia dinaminė nuokrypa (procentais) pasirinkama tik tada, kai mažinamas pereinamojo proceso laikas.
7) Optimizavimo kriterijus. Galima pasirinkti kriterijus: pereinamojo proceso trukmės mažiausią vertę, didžiausios dinaminės nuokrypos mažiausią vertę arba modifikuotą integralinį kriterijų.
8) Paieškos pobūdis. Didžiausios arba mažiausios paieška.

Sistemos pradinio pereinamojo proceso grafikas, kai pradinis poveikis $u(t) = const = 1$ pateiktas **5.2 pav**. Uždaros kvazioptimalios valdymo sistemos sintezės uždavinys sprendžiamas taikant simpleksinės paieškos algoritmus ir naudojant programų paketą *Kvazio1*. Sintezė atliekama taip: į sudaryto modelio įėjimą ateina programiškai suformuotas įėjimo signalas, kurio dydis kiekvieno bandymo metu skiriasi.

Keičiant valdymo trukmę reikėjo pasiekti geriausią (optimaliausią) pereinamąją procesą. Didinant bei mažinant valdymo trukmės vertę, buvo nustatyta jog optimali valdymo trukmės vertė yra 2,7 sekundės. Pereinamojo proceso ir poveikio grafikai yra pateikti **5.5** pav. ir **5.4** pav. atitinkamai.



4 pav. Artimas optimaliam valdymo poveikio dėsnis




**5.5 pav**. Objekto perinamasis procesas, atitinkantis valdymo dėsnį, parodytą **5.4 pav**.

Keičiant sistemos valdymo poveikio laiką galima pagerinti sistemos pereinamojo proceso parametrus – optimizuoti pereinamąjį procesą. Poveikis turi gesinti pereinamojo proceso švytavimus, tokiu būdu jį pagreitinant. Pradinio pereinamojo proceso laikas buvo 45 sekundės, o optimizuoto  tik 25 sek.

# 6. IŠVADOS IR PASIŪLYMAI

1. Robotų valdymo optimizavimas labai padidina jų panaudojimo efektyvumą. Problemų ir optimalaus valdymo metodų sprendimai skirtingų tipų robotams, nors ir turi savo specifiką, bet daugeliu atvejų labai panašus.
2. Valdant robotą pagal kelias koordinates vienu metu, jo judėjimas tampa žymiai dinamiškesnis. Jeigu nėra apribojimų ir kliūčių, robotą galima valdyti pagal visas koordinates iš karto. Toks valdymas yra žymiai efektyvesnis ir galima pasiekti optimaliausią valdymo laiką.
3. Remiantis sukurta paprograme galima sudaryti reikiamą roboto judėjimo valdymo programą , neaprašant kiekvienos judėjimo krypties atskirai. Užtenka pasirinkti reikiamas koordinates, o tai labai supaprastina roboto valdymą.
4. Optimalaus valdymo sintezė, taikant maksimumo principą, suteikia galimybę surasti mažiausią nagrinėjamo proceso laiką.
5. Programa Kvaziol galima spręsti uždavinius, kai objekto matematinis modelis sudėtingas ar jo išvis nėra ir kai klasikinių sintezės metodų taikyti neįmanoma.
6. Keičiant atviros kvazioptimalios sistemos valdymo poveikio laiką, galima pagerinti sistemos pereinamojo proceso parametrus – optimizuoti pereinamąjį procesą. Poveikis turi slopinti pereinamojo proceso švytavimus, tokiu būdu jį pagreitindamas.
7. Automatinio programavimo algoritmai užtikrina robotų adaptacija prie aplinkos ir leidžia valdyti robotus, veikiančius neapibrėžtoje ir ekstremalioje aplinkoje, kuri pavojinga žmonių gyvybei.

# 7. LITERATŪRA


1. Aleksa V., Dailidė S., Gamybos procesų automatizavimas. Vilnius: Mokslas, 1976. 343p.
2. Bulovas V. Mikroprocesoriai. Vilnius: Mokslas, 1989. 198p.
3. Dambrauskas A., Šulskis D. Kintamos struktūros valdymo sistemų modeliavimo programinė įranga // Automatika ir valdymo technologijos-99. Kaunas: Techologija, 1999, p 22-24.
4. Dambrauskas A., Šulskis D. Kvazioptimalių ir kintamos struktūros sistemų sintezės programinė įranga. Mokomoji knyga. Vilnius, Technika, 2002. 63p.
5. Dambrauskas A. Simpleksinės paieškos metodai.Vilnius: Technika, 1995. 230p.
6. Dambrauskas A. Valdymo objektų optimizacija ir indetifikacija. Vilnius: Technika, 1994. 29p.
7. Elektrotechnikos terminų žodynas. Kaunas.: Technologija, 1999. 871 p.
8. Elektrotechnikos terminų žodynas : ADIAFORA [diskas]. Kauno technologijos universitetas, 2001.
9. Ezerskas N. Modicon programuojamieji valdikliai. Kaunas: Technologija, 1996. 286p.
10. Mogilnickas I., Petrauskas. R. Robotų technika. Vilnius: VPI, 1990. 94p.
11. Paknienė F. Pramoniniai robotai ir manipuliatoriai. Vilnius: KaunoAntano Sniečkaus politechnikos institutas, 1982. 44p.
12. Poška A. Tipinių technologinių procesų ir įrenginių automatizavimas. Vilnius: Mokslas, 1992. 248p.
13. Pramoninia robotai. Technologinės galimybės ir techninės charakteristikos. Kaunas: Kauno Antano Sniečkaus politechnikos institutas, 1988. 66p.
14. Modicon 311/411 Micro PLC Hardware User Manual. MODICON, Inc. Industrial Automatikon Systems – North Androver, Massachusets, 1993. 46p.

15. Modicon Compact 984 Ladder Logic Manual. Modicon, Inc., Industrial Automation System. – North Androver, Massachusets March, 1993. 95p.
16. Modicon Modsoft Lite Programmer User Manual. Modicon, Inc., Industrial Automation System. – North Androver, Massachusets March, 1993. 104p.
17. Робототехника и гибкие автоматизированные производств., Кн.2 / Под редакцией Макарова И.М., / - Москва: Высшая школа, 1986. 175p.
18. Управление манипуляционными роботами / М. Вукобратович, Д. Стокич/- М.: Наука, 1985. 370p.
19. Управляющие системы промышленных роботов / Ю. Д. Андрианов, Л. А .Глейзер и др. Под редакцией И. М. Макарова./ – М.: Машиностроение, 1984. 228p.
20. ПЛК Modicon [interaktivus] 2003 [žiūrėta 2003-11-04] Prieiga per internetą: http://www.indusoft.ru/fanuc/products.html
21. Краткие сведения о системе автоматизированной разработки программ для УРТК [interaktivus] 2000 [žiūrėta 2002-09-15] Prieiga per internetą: http://robot.kgtu.runnet.ru/URTK.htm
22. Программируемые контроллеры TSX Compact [interaktivus] 2002 [žiūrėta 2003-12-10] Prieiga per internetą: http://www.modicon.ru
23. Программируемый логический контроллер Modicon [interaktivus] 2002 [žiūrėta 2003-12-10] Prieiga per internetą: http://www.sotrudnik.perm.ru
24. Средства и системы автоматизации [interaktivus] 2002 [žiūrėta 2004-01-15] Prieiga per internetą: http://www.rtsoft.ru/products/Modicon-TSX-Quantum/
25. Универсальный РоботоТехнический Комплекс [interaktivus] 2003 [žiūrėta 2004-01-20] Prieiga per internetą: http://urtk.nm.ru/index.html

Vilniaus gedimimo technikos universitetas
Elektronikos fakultetas
Automatikos katedra

ISBN ISSN
Egz. sk.
Data

Magistratūros baigiamasis darbas (tezės)

Pavadinimas: Optimization of robot control system

Autorius: Jaroslavas Semaško, gr. AUSm-2

Kalba

☐ Lietuvių

☐ Užsienio

**Summary**

In final master thesis there are described robots' control optimization methods and control algorithms.

In review there are discussed automated control systems of robots and control of robots, compatibility of robot and microcontroller, problems of search optimization.

The main master thesis aim is to optimize robot's control system by using search optimization methods. An algorithm of URTK robot was optimized and new subroutine of robot control system was created. There was optimized transient process of robot control open system, using Kvazio1 software and its algorithmic methods of syntheses.

Using results of research and there are made conclusions and suggestions.

.